Verification of Concurrent Programs under Weakly Consistent Models

Ahmed Bouajjani

Université Paris Cité

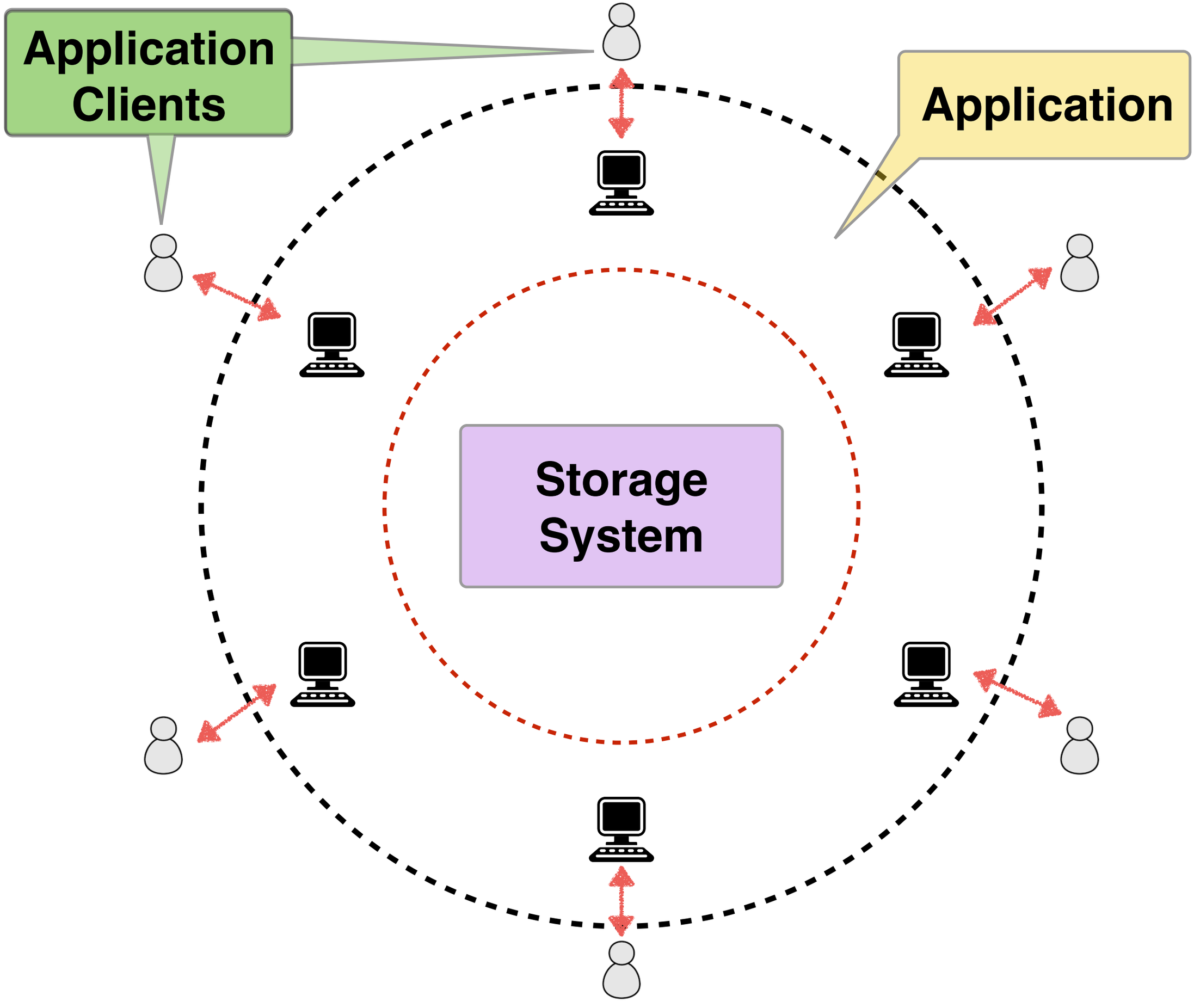
Mohamed Faouzi Atig Sebastian Burckhardt Madan Musuvathi

Roland Meyer Egor Derevenetc

Parosh Aziz Abdulla Tuan Phong Ngo

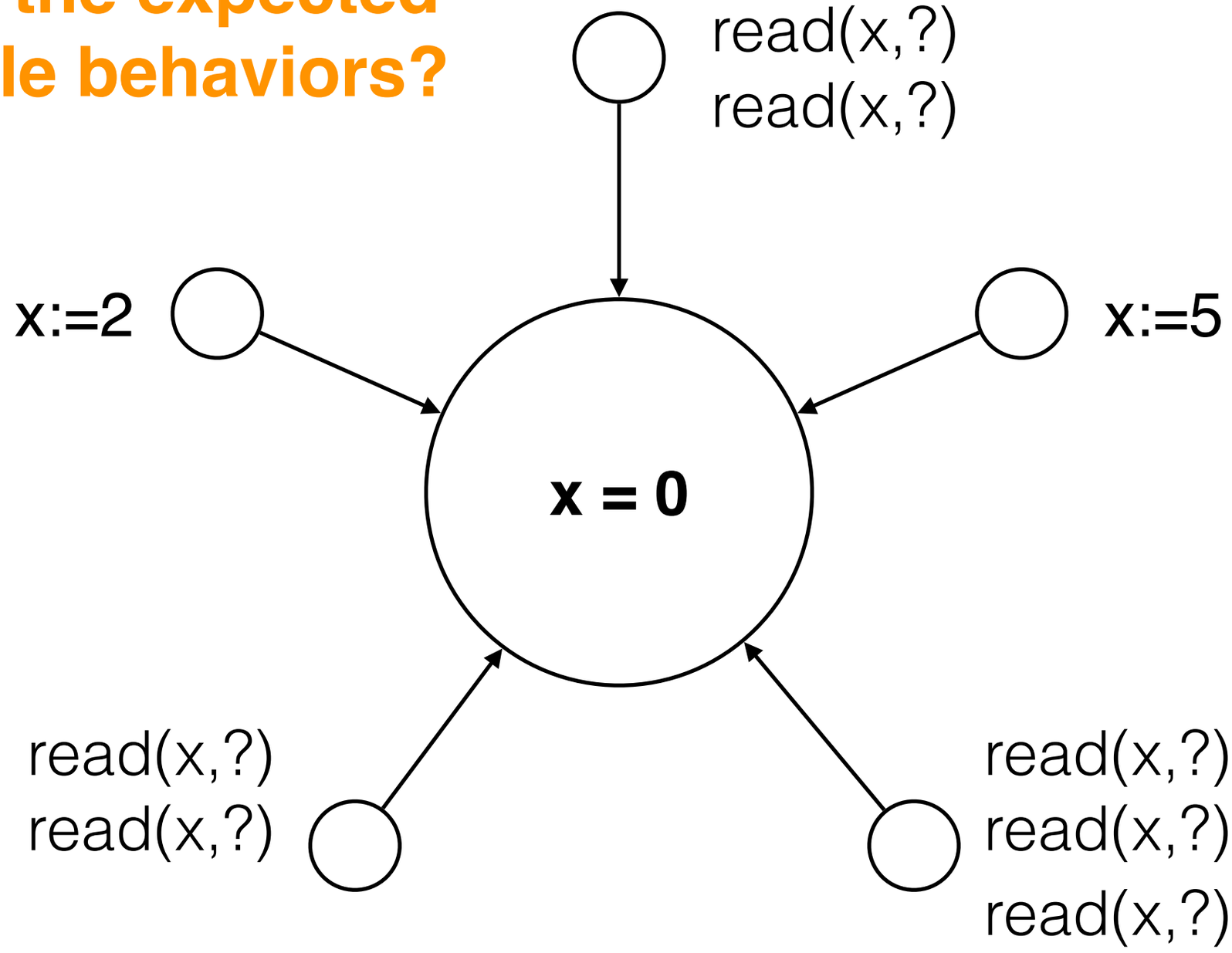
Constantin Enea Sidi Mohammed Beilahi

CONCUR, Antwerp, Belgium, 2023


Application Clients
Application
Storage System

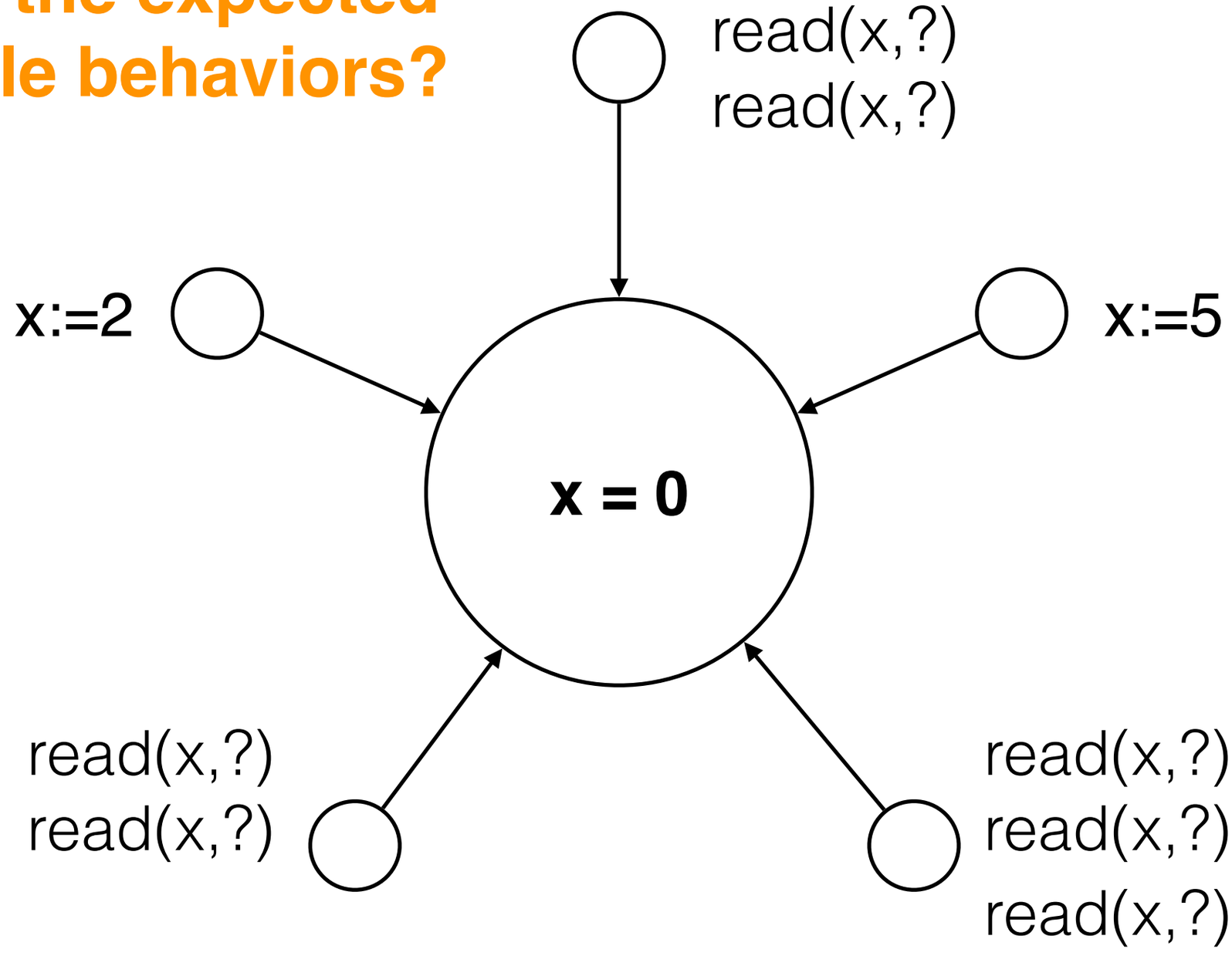
Interactions with a memory

What are the expected observable behaviors?



Interactions with a memory: visibility

What are the expected observable behaviors?

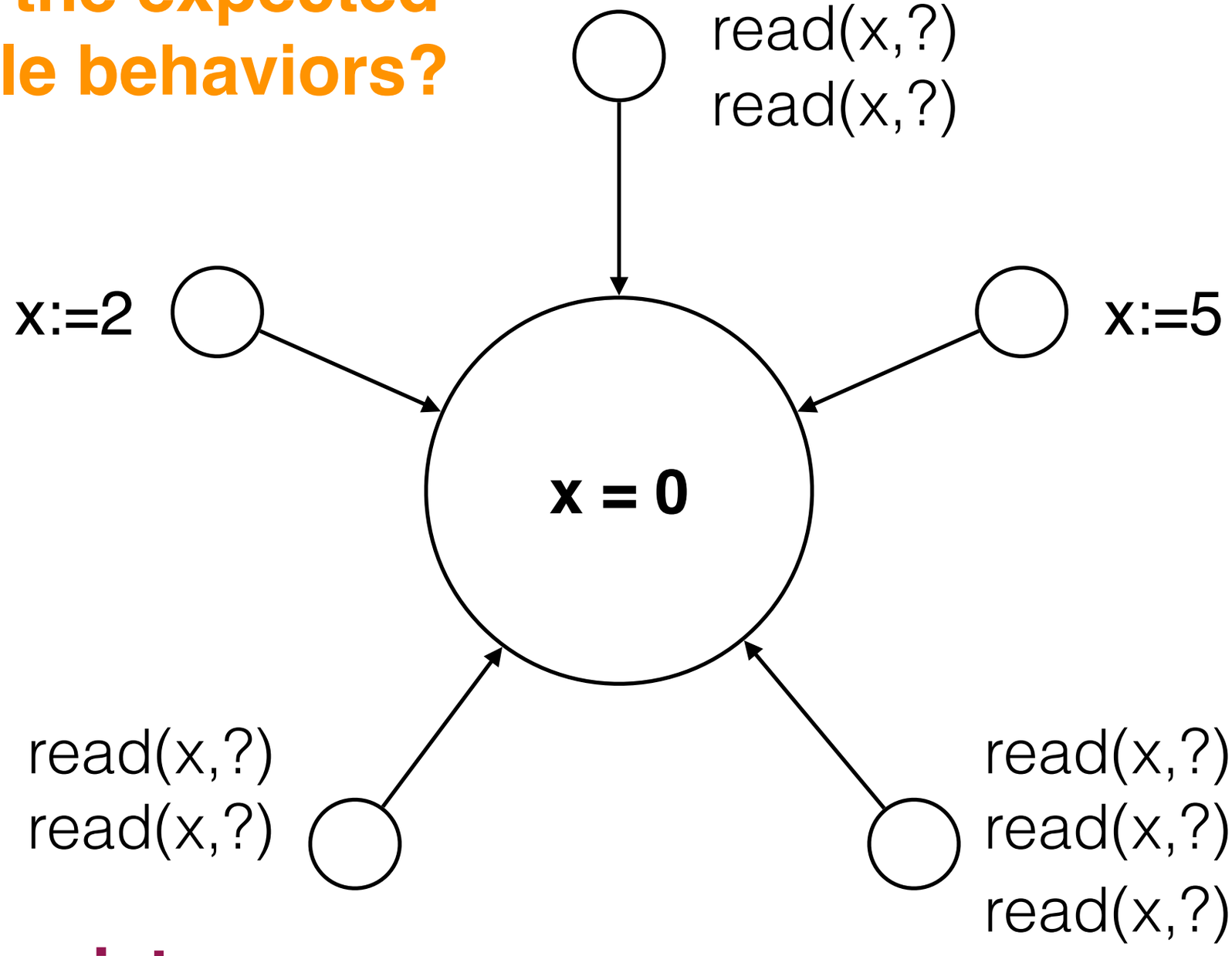


Returned values by read actions depend on:

- the current set of visible actions by each process, and
- the order in which actions are seen by each process

Interactions with a memory: Strong Consistency

What are the expected observable behaviors?

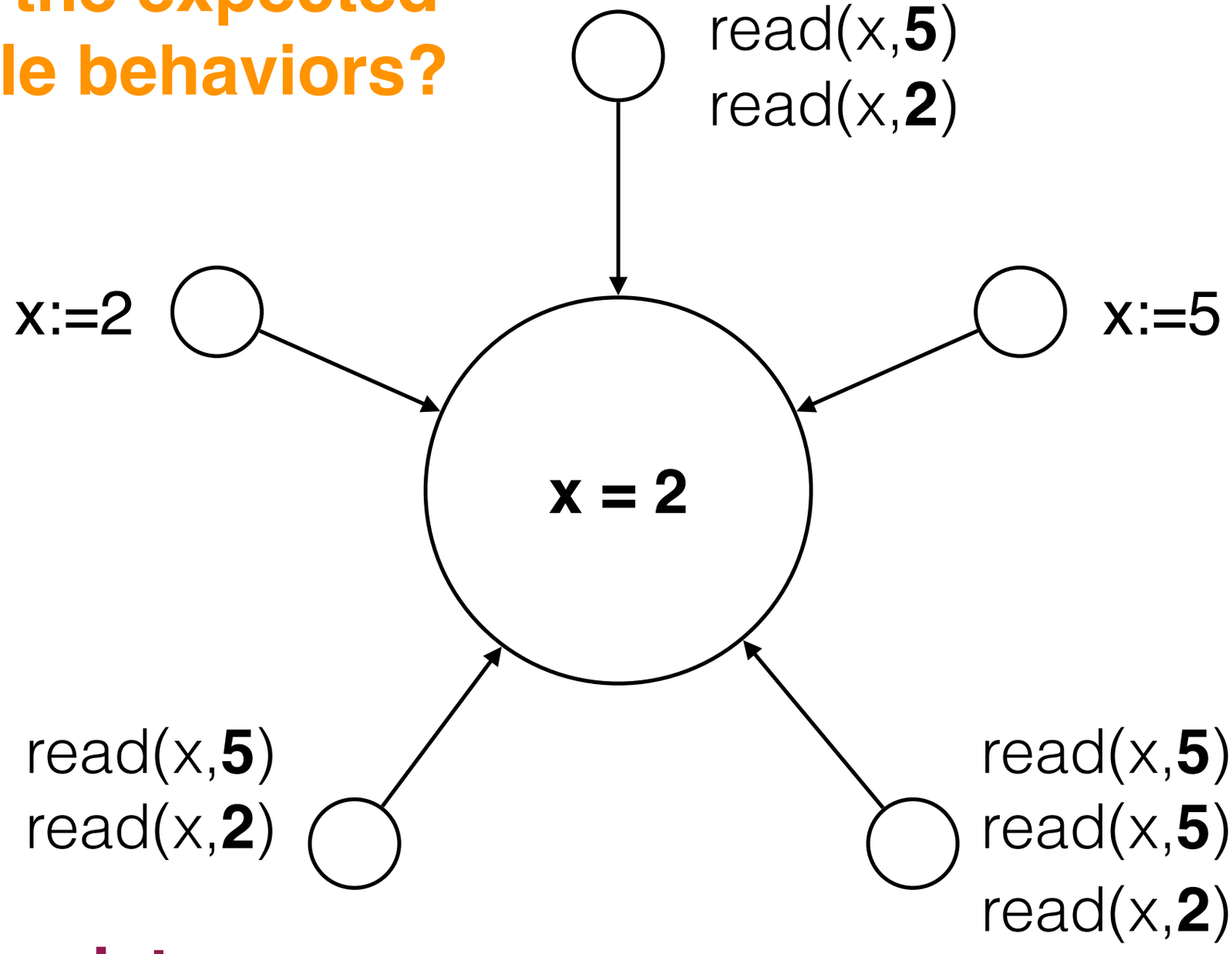


Strong consistency:

- **updates are visible to all participants without delay**
- **updates are visible in the same order to everybody**

Interactions with a memory: Strong Consistency

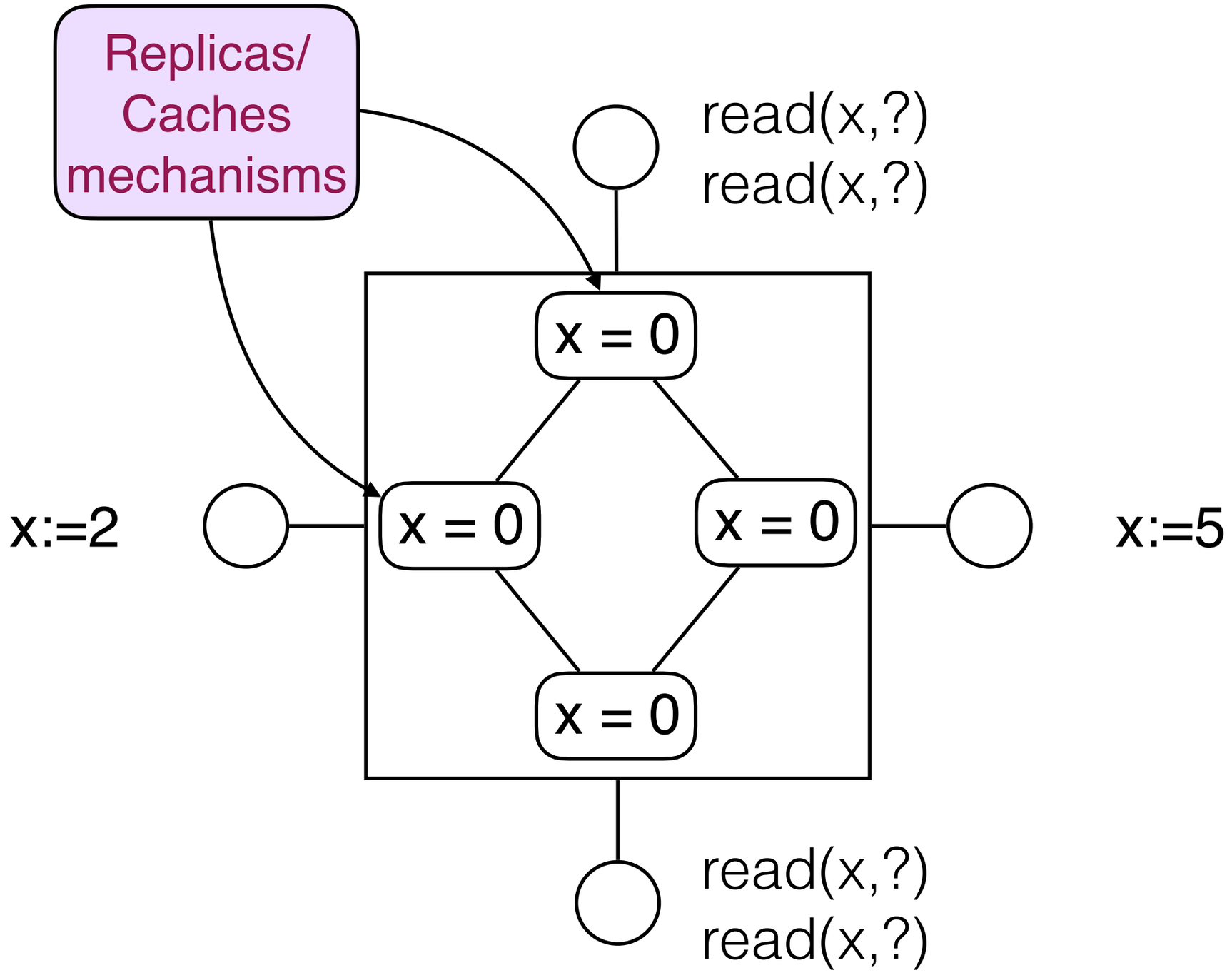
What are the expected observable behaviors?



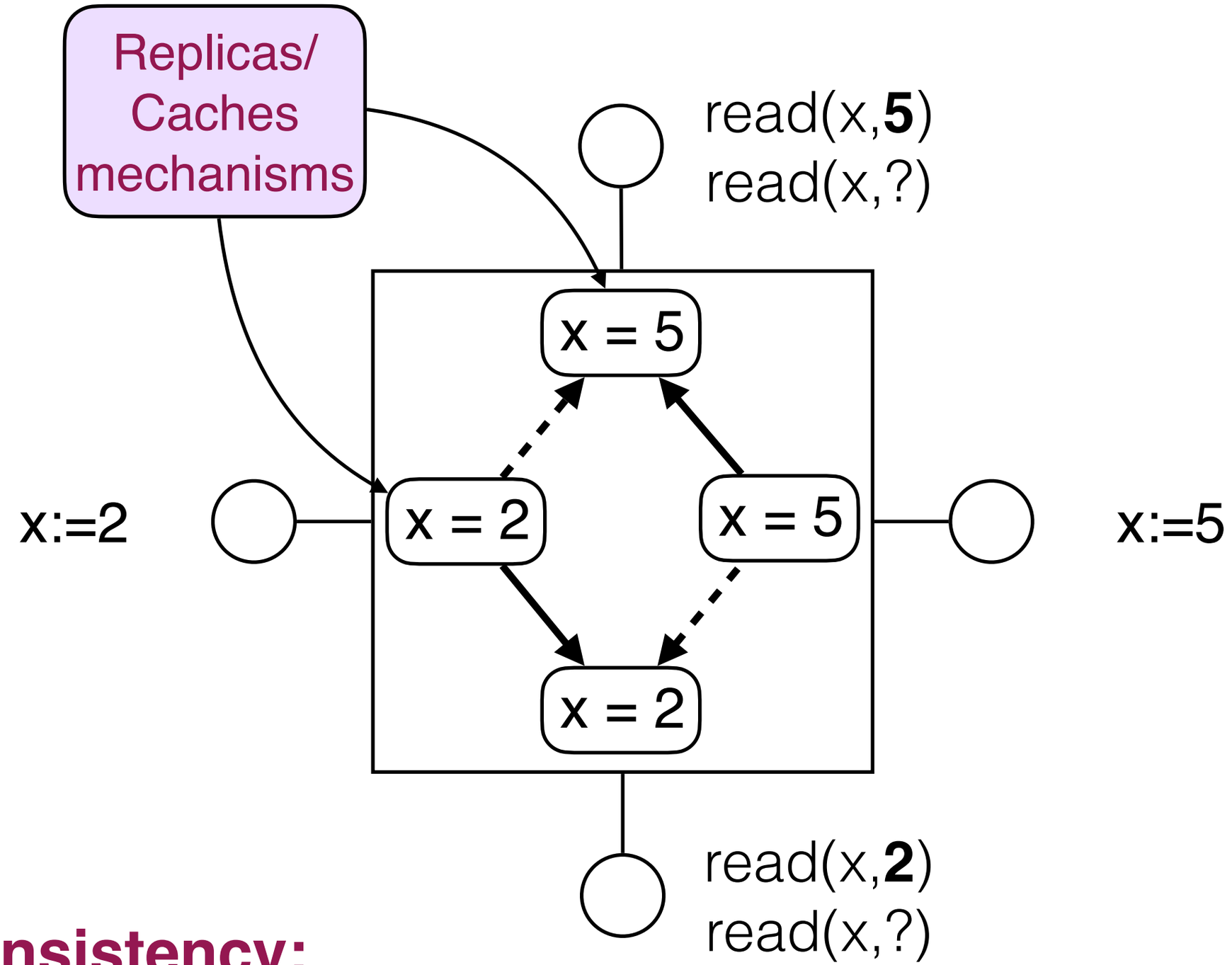
Strong consistency:

- **updates are visible to all participants without delay**
- **updates are visible in the same order to everybody**

Interactions with a memory



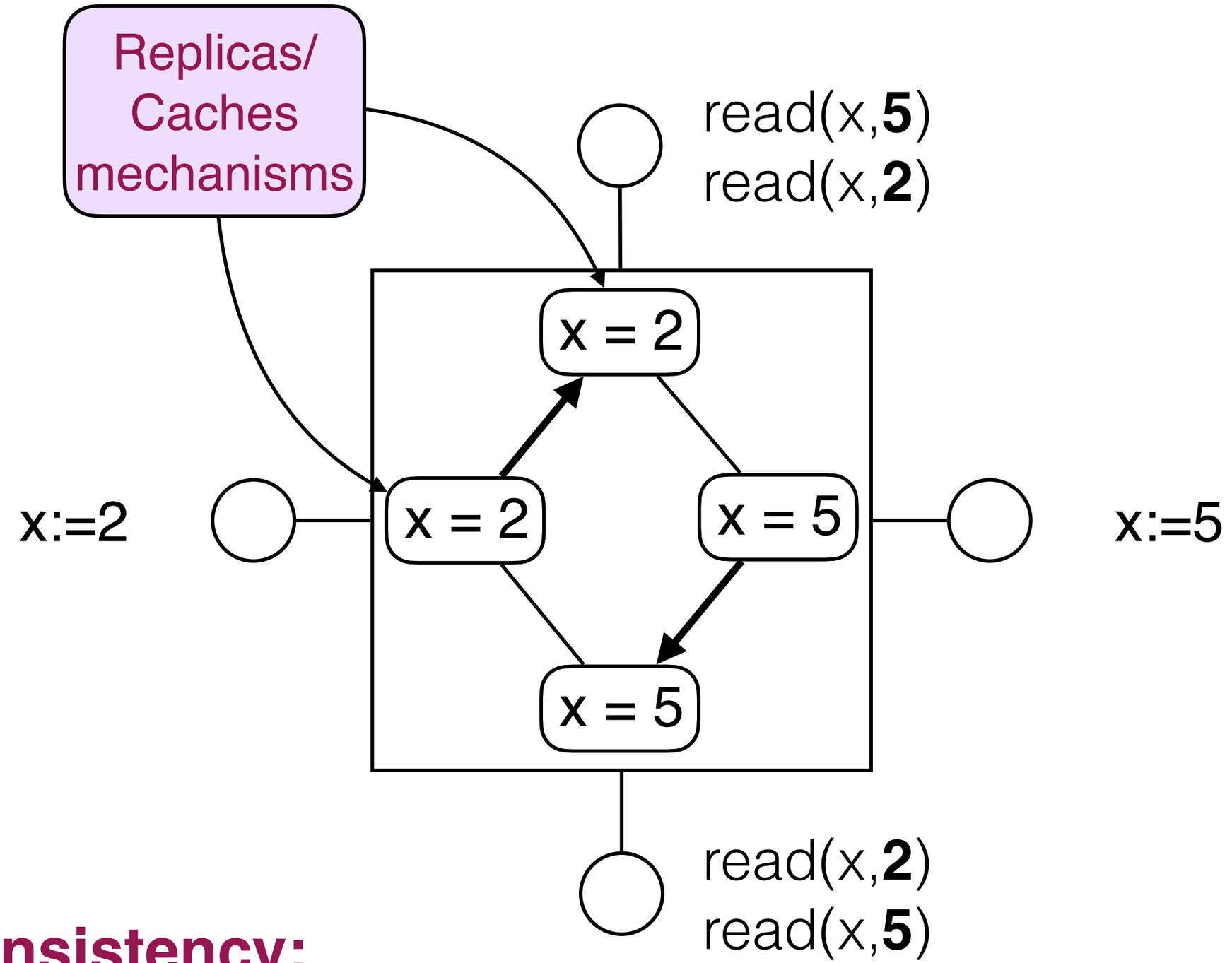
Interactions with a memory



Weak consistency:

- participants may **see different sets of updates**

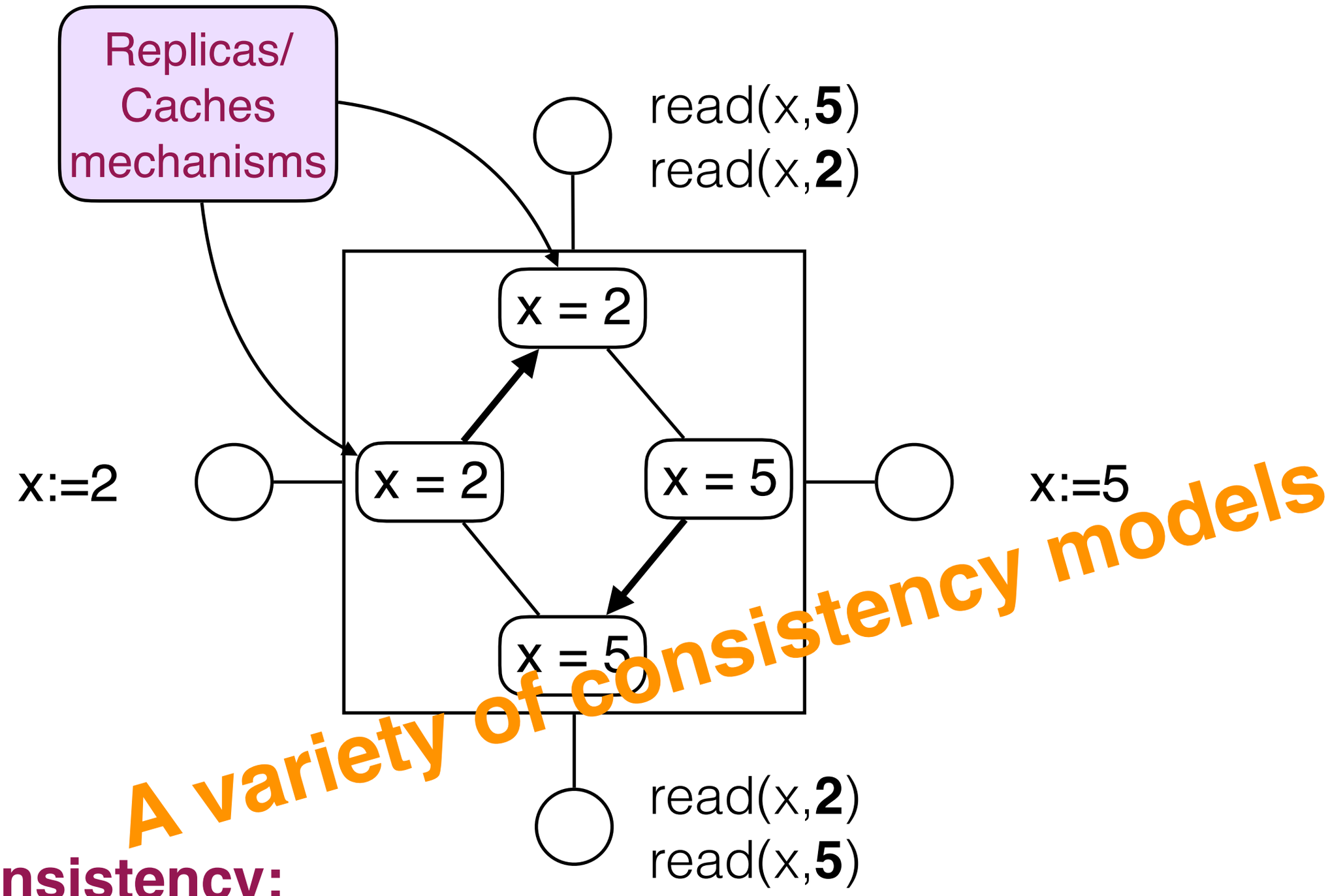
Interactions with a memory



Weak consistency:

- participants may **see different sets of updates**
- updates may be **visible in different orders** to participants

Interactions with a memory



A variety of consistency models

Weak consistency:

- participants may **see different sets of updates**
- updates may be **visible in different orders** to participants

Sequential Consistency

Lamport 79

Operational semantics:

Interleaving of actions of the different processes

Sequential Consistency

Lamport 79

Operational semantics:

Interleaving of actions of the different processes

Axiomatic semantics:

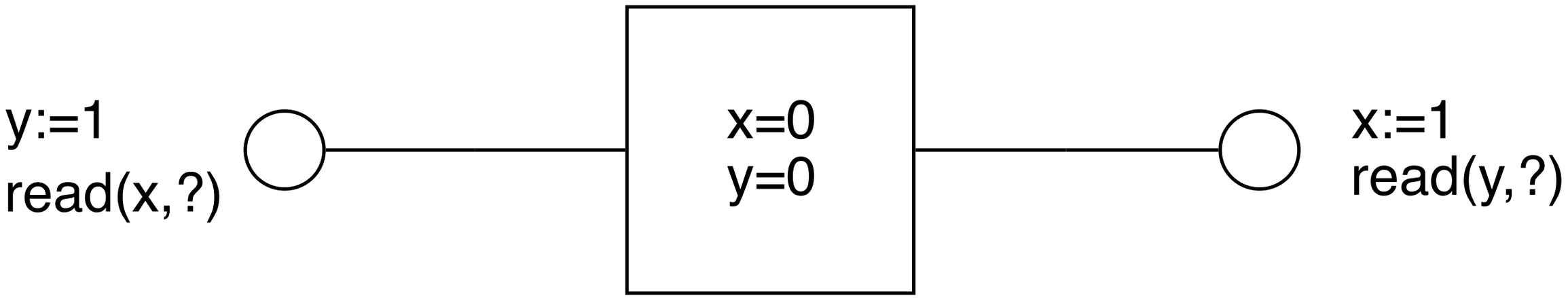
- **rf** (read-from): write is the source of a read
- **so** (store-order): total order between updates
- **po** (program-order): order between operations in a same process
- **cf** (conflict): reads happen-before conflicting writes

$$\frac{w(x, u) \text{ — so —> } w(x, v) \qquad w(x, u) \text{ — rf —> } r(x, u)}{r(x, u) \text{ — cf —> } w(x, v)}$$

hb (happen-before) = **union of rf, so, po, and cf**, is ***acyclic***

Sequential Consistency

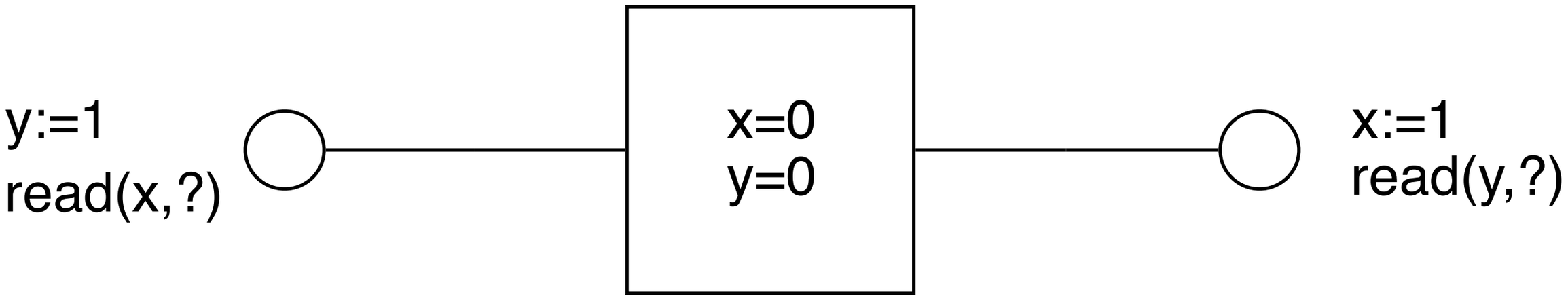
Lamport 79



- **updates are totally ordered** => visible in the same order to all proc.
- **program order is respected** => e.g., reads cannot overtake writes

Sequential Consistency

Lamport 79

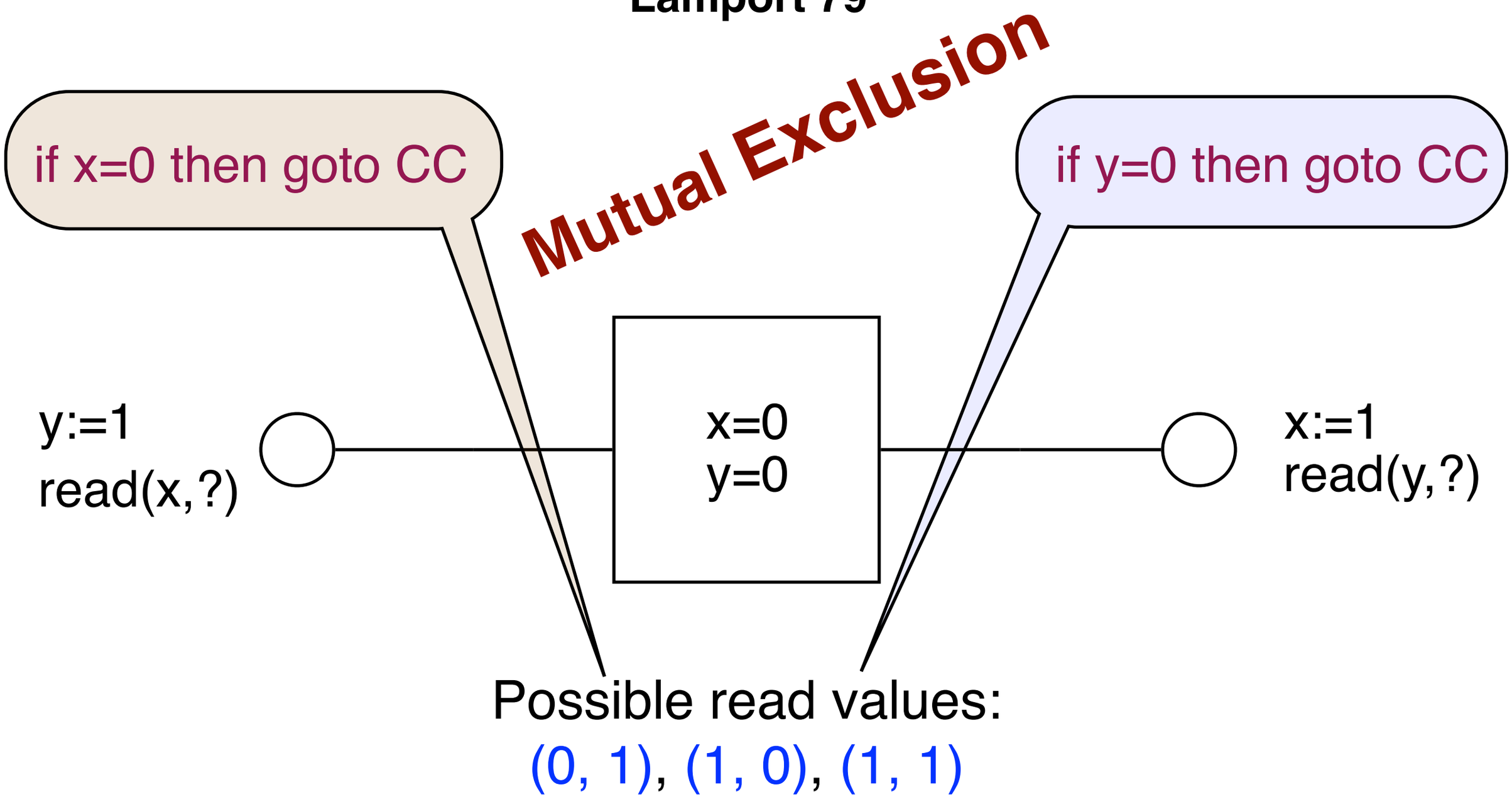


Possible read values:
(0, 1), (1, 0), (1, 1)

- **updates are totally ordered** => visible in the same order to all proc.
- **program order is respected** => e.g., reads cannot overtake writes

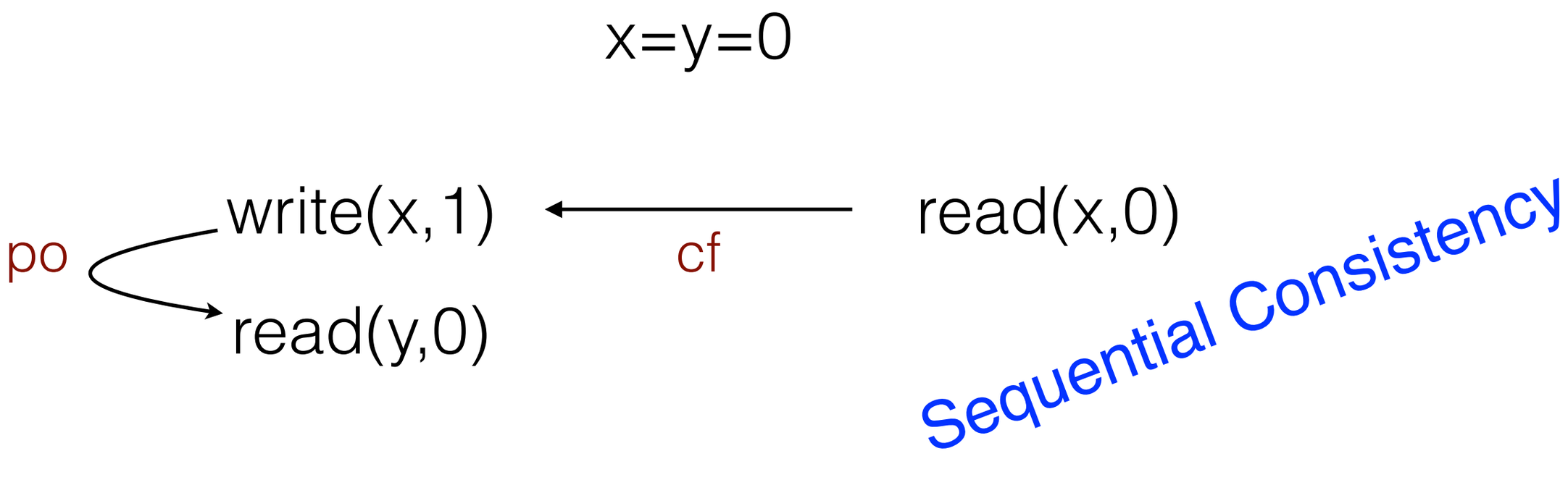
Sequential Consistency

Lamport 79



- **updates are totally ordered** => visible in the same order to all proc.
- **program order is respected** => e.g., reads cannot overtake writes

Relaxing order constraints



read(x,0) write (x,1) read(y,0)

Relaxing order constraints

Relax the Program Order Constraints

x=y=0

write(x,1) ← cf — read(x,0)

po: write(x,1) → read(y,0)

~~Sequential Consistency~~

read(x,0) write (x,1) read(y,0)

Swap operations

read(x,0) read(y,0) write (x,1)

Relaxing order constraints

Relax the Program Order Constraints

x=y=0

write(x,1) ← cf — read(x,0)

po: write(x,1) → read(y,0)

~~Sequential Consistency~~

read(x,0) write (x,1) read(y,0)

Swap operations

read(x,0) read(y,0) write (x,1)

Execute in parallel

Fast execution of reads!

Relaxing order constraints

Relax the Program Order Constraints

x=y=0

write(x,1) ← cf — read(x,0)

po: write(x,1) → read(y,0)

~~Sequential Consistency~~

read(x,0) write (x,1) read(y,0)

Swap operations

read(x,0) read(y,0) write (x,1)

Reach the same state!

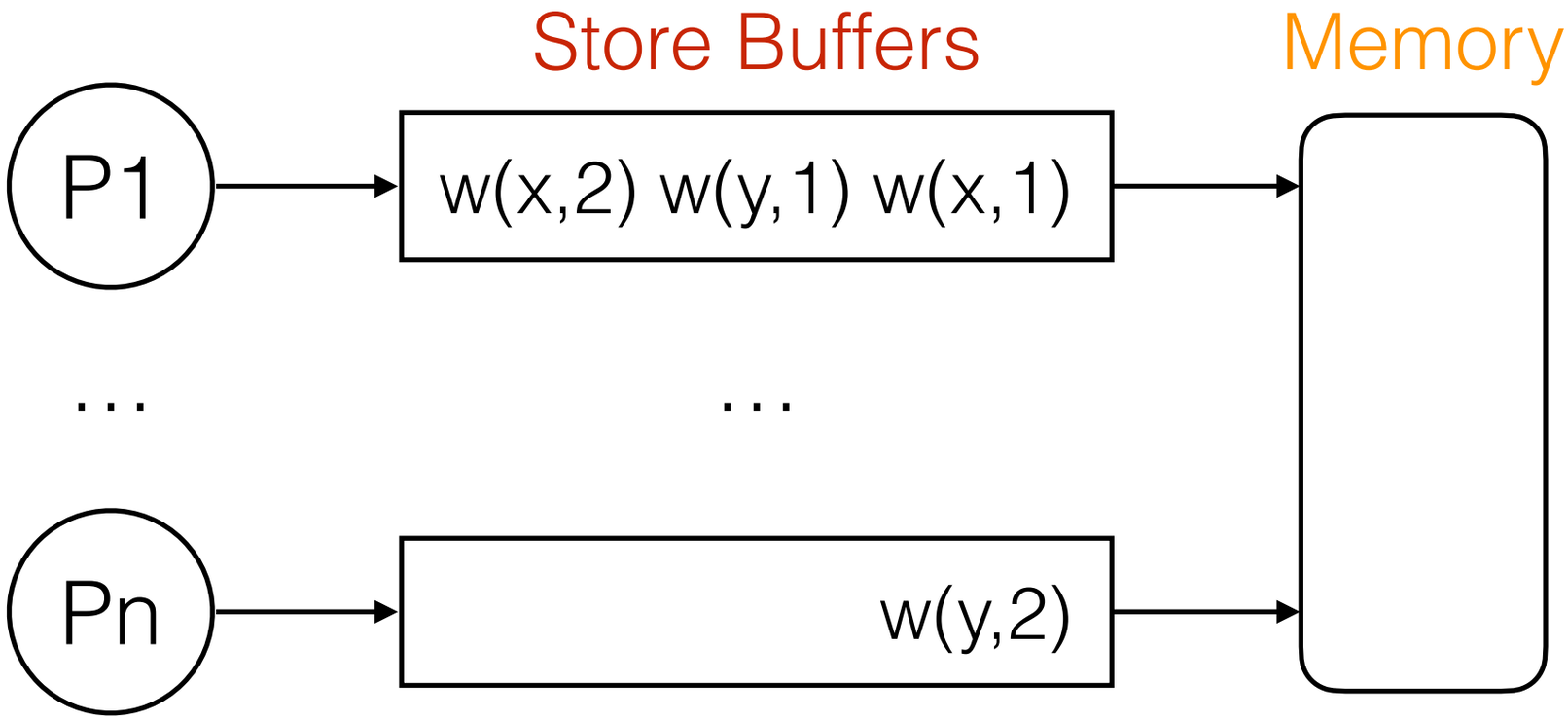
Execute in parallel

Fast execution of reads!

Weak Consistency Models

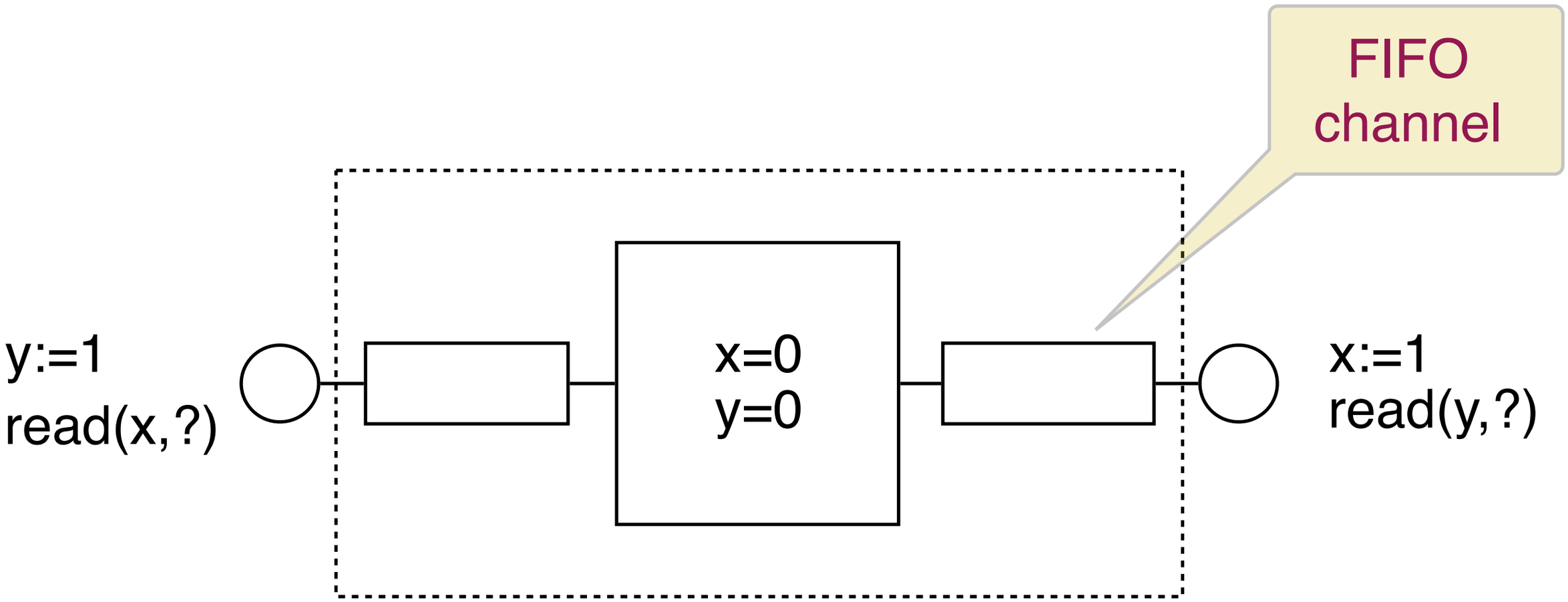
- Complex program semantics
- Reordering of operations, unbounded forward/backward moves
- Operational semantics: State machines + unbounded queues

TSO : Operational Model



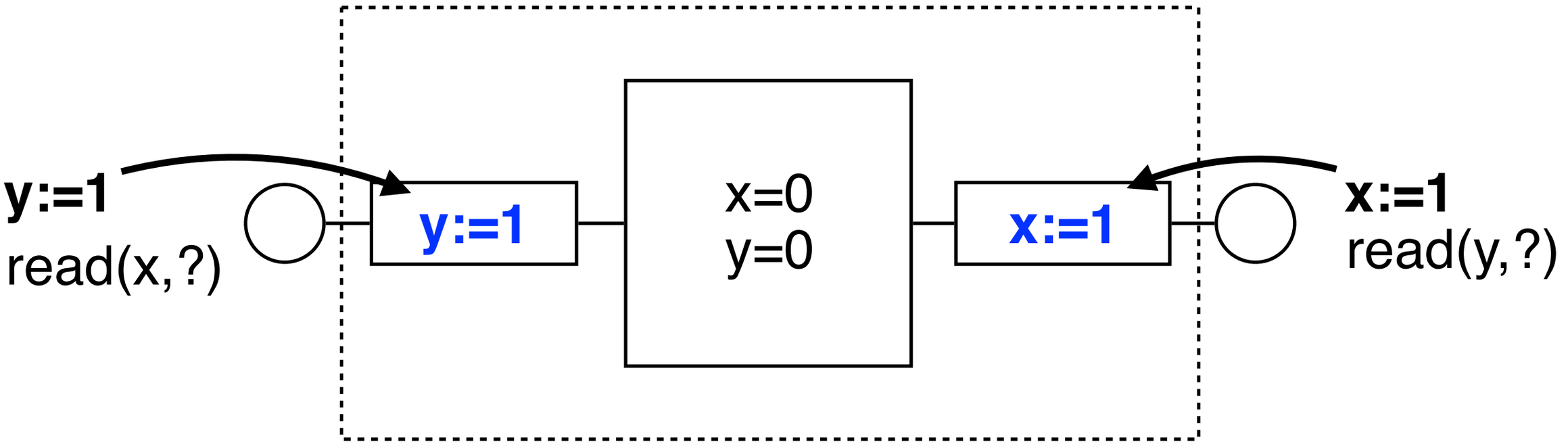
- writes are sent to store buffers (one per process)
- writes are committed to memory at any time
- reads are from
  - **own store buffer** if a value exists (last write to the variable)
  - otherwise from the **memory**
- atomic read-writes executed when **own buffer is empty**
- fence = flush the buffer (simulated with atomic read-write)

Total Store Ordering (TSO)



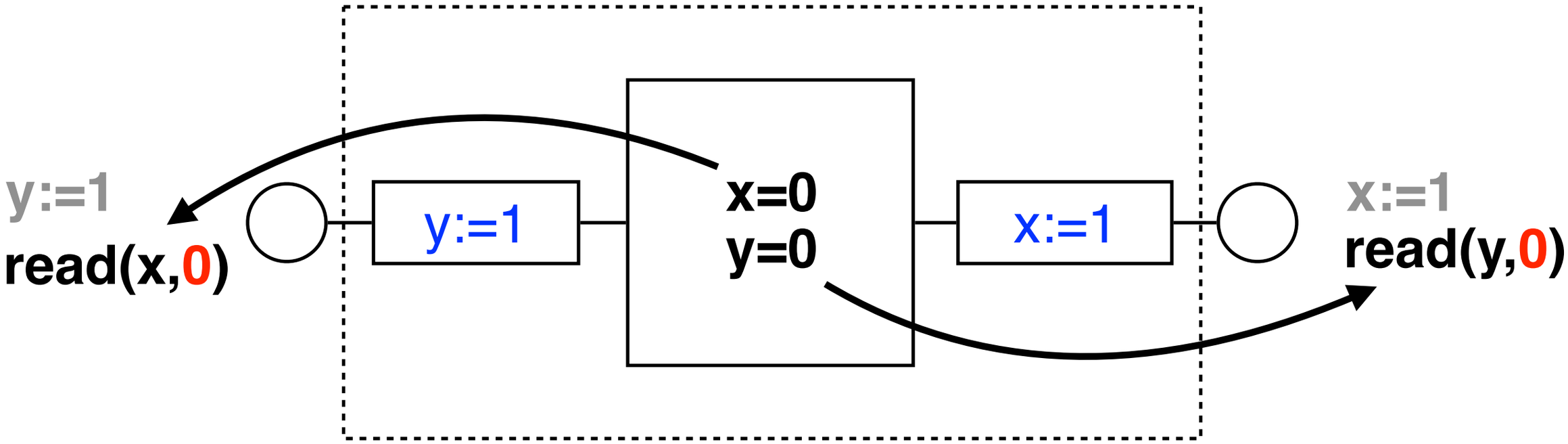
- updates are totally ordered => visible in the same order to all proc.,
- updates can be delayed => reads may overtake writes

Total Store Ordering (TSO)



- updates are totally ordered => visible in the same order to all proc.,
- updates can be delayed => reads may overtake writes

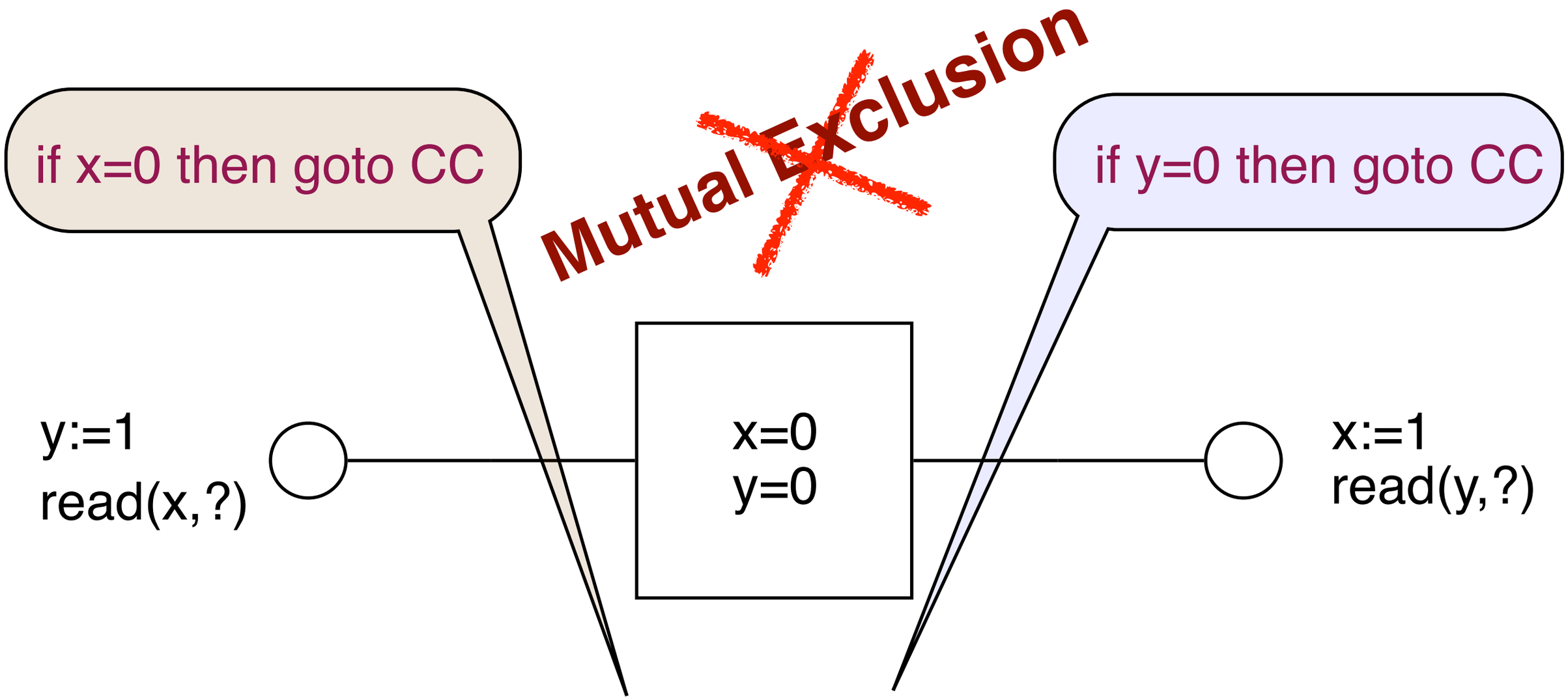
Total Store Ordering (TSO)



It is also possible to read (0, 0)

- updates are totally ordered => visible in the same order to all proc.,
- updates can be delayed => reads may overtake writes

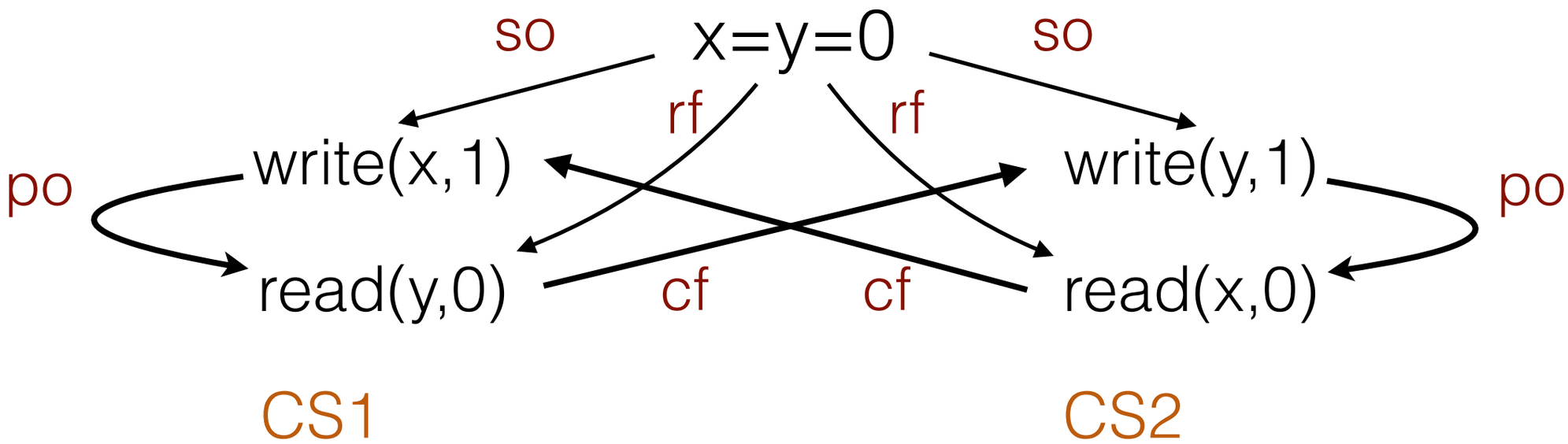
Total Store Ordering (TSO)



Possible read values:
(0, 1), (1, 0), (1, 1), **(0, 0)**

- updates are totally ordered => visible in the same order to all proc.,
- updates can be delayed => reads may overtake writes

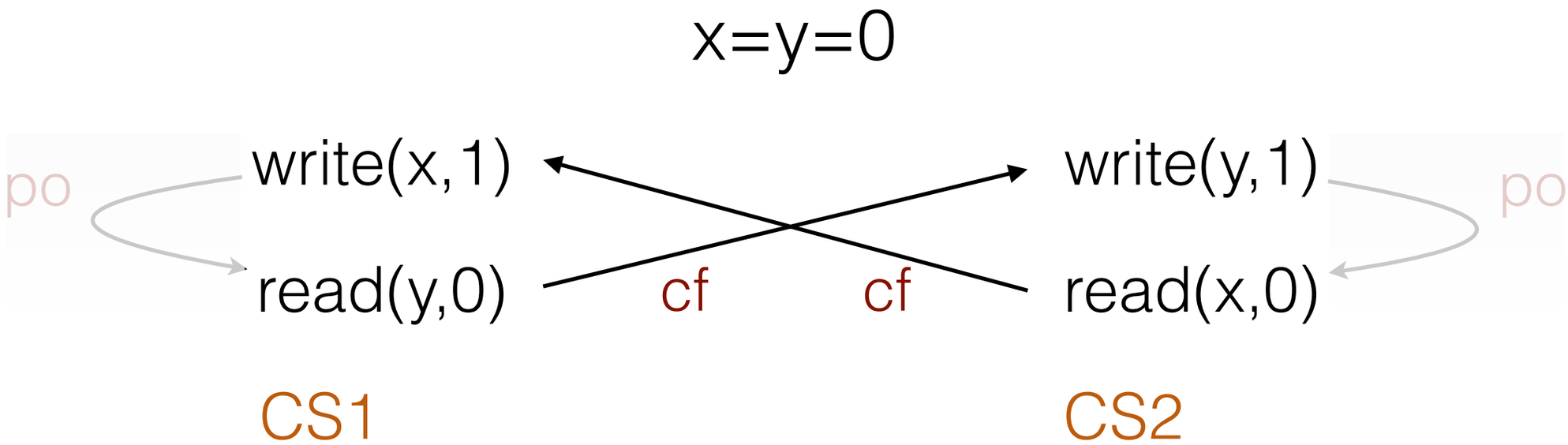
TSO: Non SC Behaviors



CS1 and CS2 ?

- **Impossible under SC**: Cyclic happen-before relation

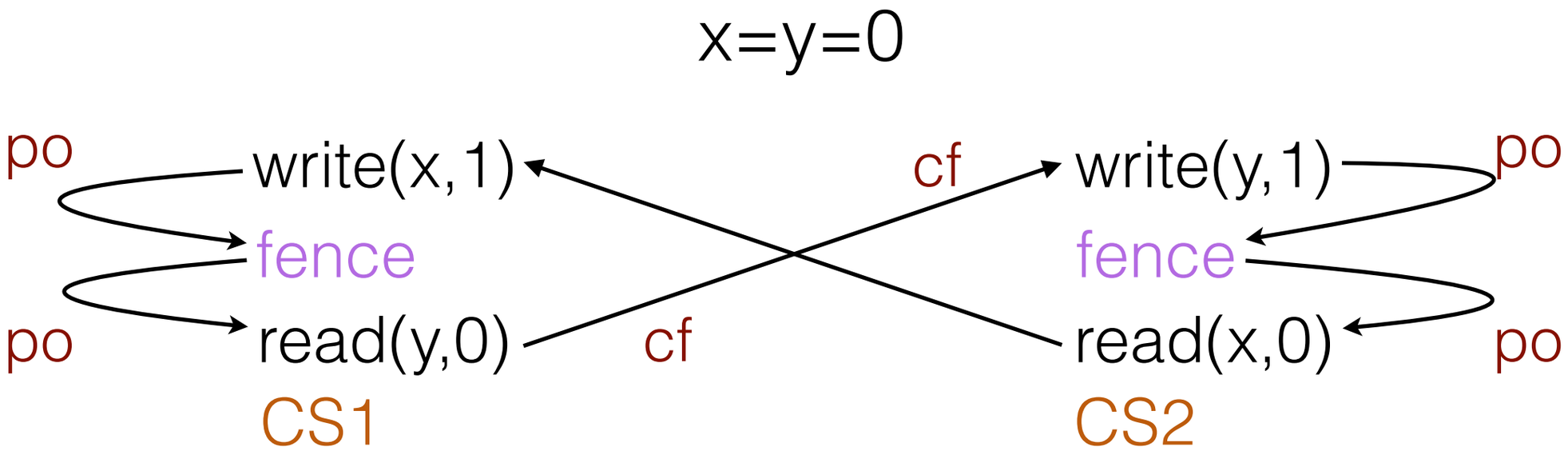
TSO: Non SC Behaviors



CS1 and CS2 ?

- **Impossible under SC**: Cyclic happen-before relation
- **Possible under TSO!**
  - writes are **delayed**: pending in store buffers
  - reads get old values in the memory (0's)

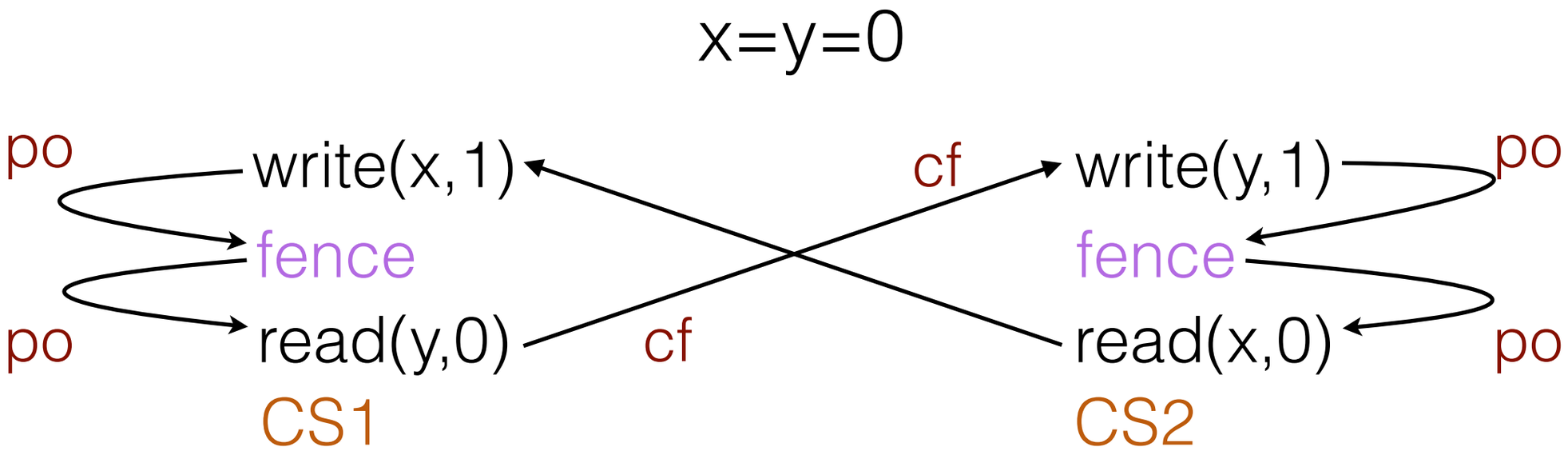
Avoiding Reordering: Fences



CS1 and CS2 ?

- A fence forces flushing the store buffer
- => reaching CS1 and CS2 becomes impossible

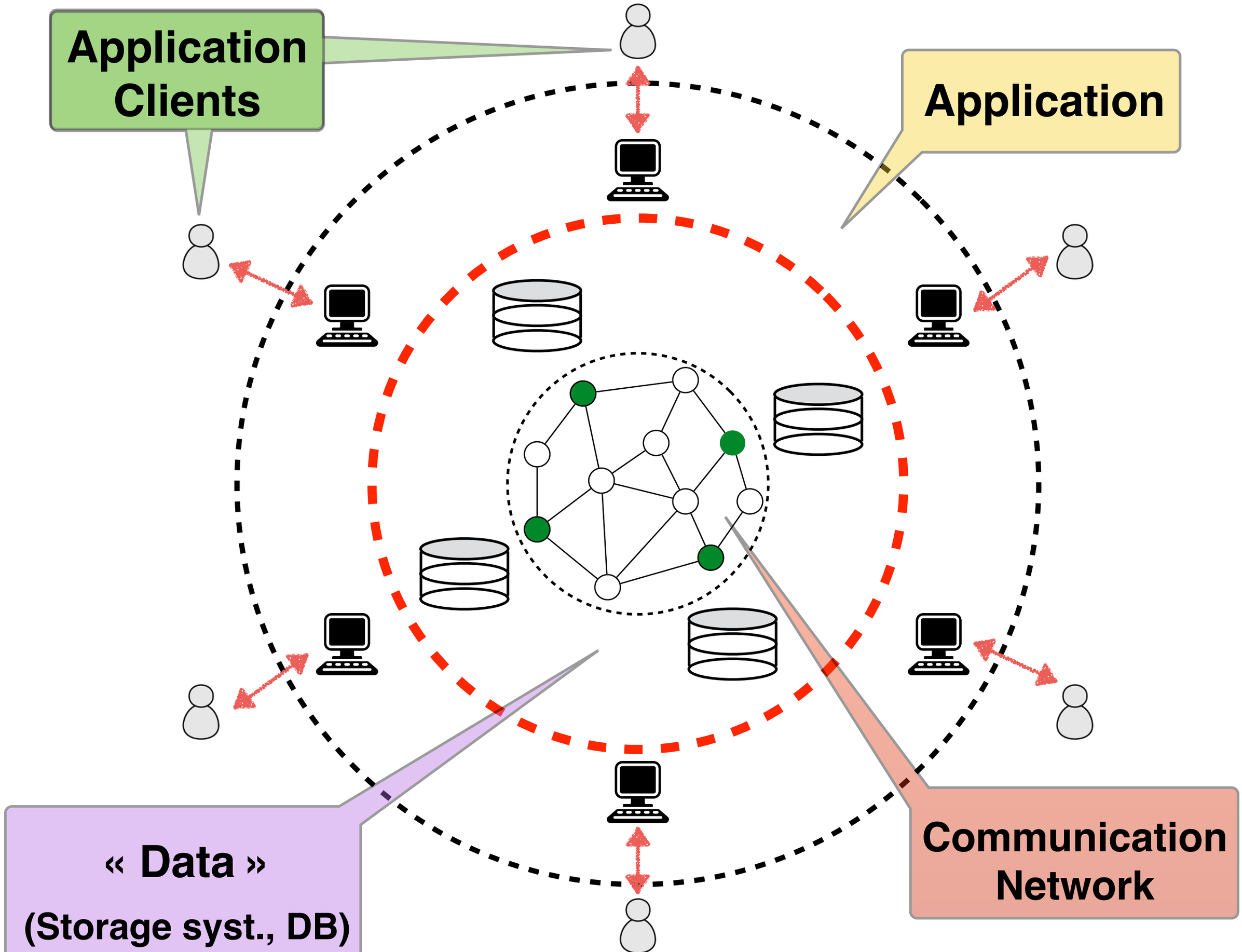
Avoiding Reordering: Fences



CS1 and CS2 ?

- A fence forces flushing the store buffer
- => reaching CS1 and CS2 becomes impossible

SC can be enforced: insert a fence after each write


Application Clients
Application
« Data »
(Storage syst., DB)
Communication Network

Reasoning under Weak Consistency

Issues

- Formal definition of consistency models

- Express constraints on the possible orders between operations
- Operational semantics

- Verify an application under a weak consistency model

Complex behaviors due to action reordering

- Verify a storage system/DB w.r.t. a consistency level

Complex implementations with synchronisation optimizations

Reasoning under Weak Consistency

Issues

- Formal definition of consistency models

- Express constraints on the possible orders between operations
- Operational semantics

- Verify an application under a weak consistency model

Complex behaviors due to action reordering

- Verify a storage system/DB w.r.t. a consistency level

Complex implementations with synchronisation optimizations

Reasoning under Weak Consistency

Issues

- Formal definition of consistency models

- Express constraints on the possible orders between operations
- Operational semantics

- Verify an application under a weak consistency model

Complex behaviors due to action reordering

- Verify a storage system/DB w.r.t. a consistency level

Complex implementations with synchronisation optimizations

- Decidability and complexity

Action reordering can lead to undecidability/high complexity

- Testing / Static Analysis

Coverage / Accuracy

Verifying Application Correctness (safety) under Weak Consistency

Decidability?

Verifying Application Correctness (safety) under Weak Consistency

Decidability?

- Reductions to reachability in **Well Structured Systems**
  - *Well quasi ordering on the state space*
  - *Monotonicity of transition relation w.r.t to the WQO*

[AKJT'96, FS '01]

Verifying Application Correctness (safety) under Weak Consistency

Decidability?

- Reductions to reachability in **Well Structured Systems**
  - *Well quasi ordering on the state space*
  - *Monotonicity of transition relation w.r.t to the WQO*

[AKJT'96, FS '01]

- **=> TSO** [Atig, B., Burkhardt, Musuvathi'10][Abdulla, Atig, B., Ngo'18]
- **=> relaxations of TSO** [Atig, B., Burkhardt, Musuvathi'12]
- **=> TSO + persistency** [Abdulla, Atig, B., Kumar, Saivasan'21]
- **=>** other models [Lahav, Boker'20]

Verifying Application Correctness (safety) under Weak Consistency

Undecidability

- TSO + writes overtake reads (speculative reads)
  [Atig, B., Burkhardt, Musuvathi'10, 12]

- Power [Abdulla, Atig, B., Derevenetc, Leonardsson, Meyer'20]

- other models [Abdulla, Arora, Atig, Krishna'19]

From TSO programs to Lossy Channel Systems

From TSO programs to Lossy Channel Systems

But store buffers are not lossy !

An example of TSO program

x=y=0

P1

> w(x,1)

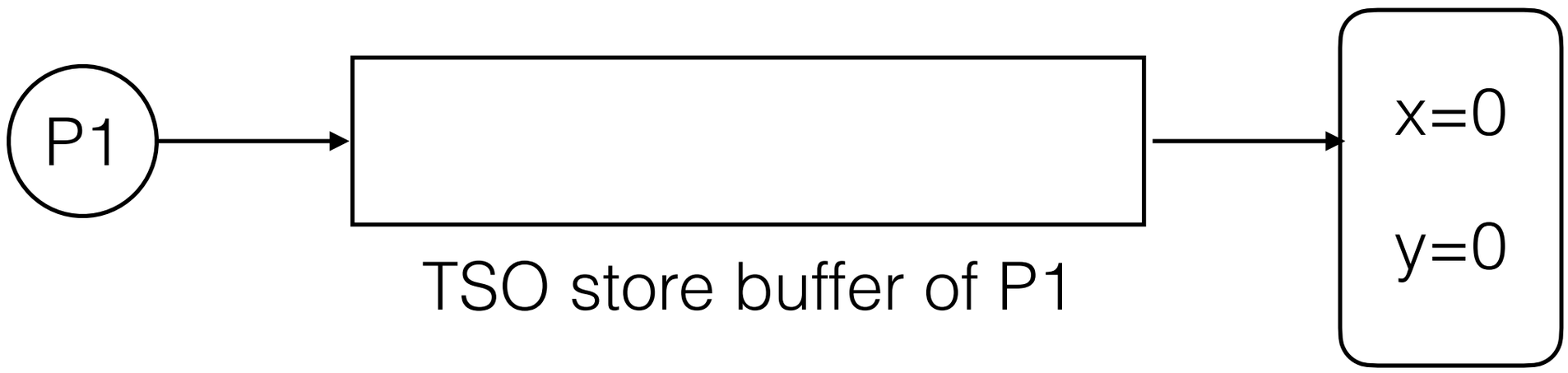
w(y,1)

w(x,2)

P2

> r(x,2)

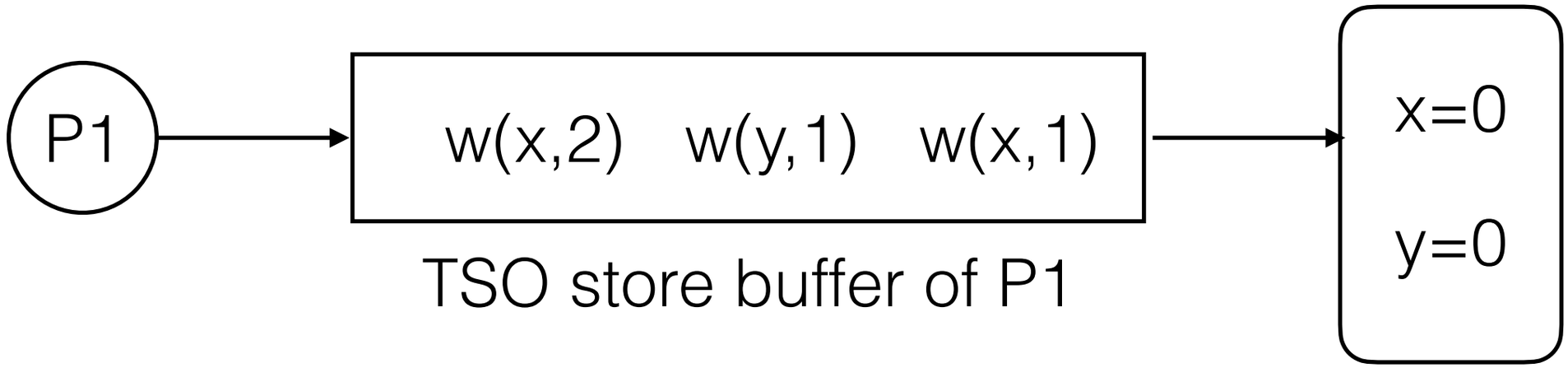
r(y,0)



An example of TSO program

x=y=0

P1	P2
w(x,1)	r(x,2)
w(y,1)	r(y,0)
w(x,2)	

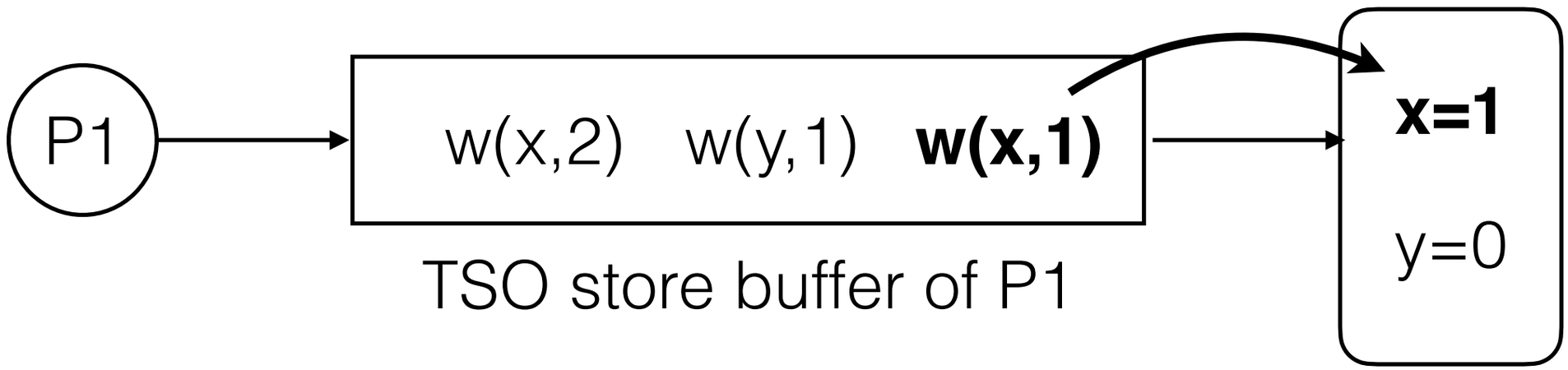


TSO store buffer of P1

An example of TSO program

x=y=0

P1	P2
w(x,1)	r(x,2)
w(y,1)	r(y,0)
w(x,2)	

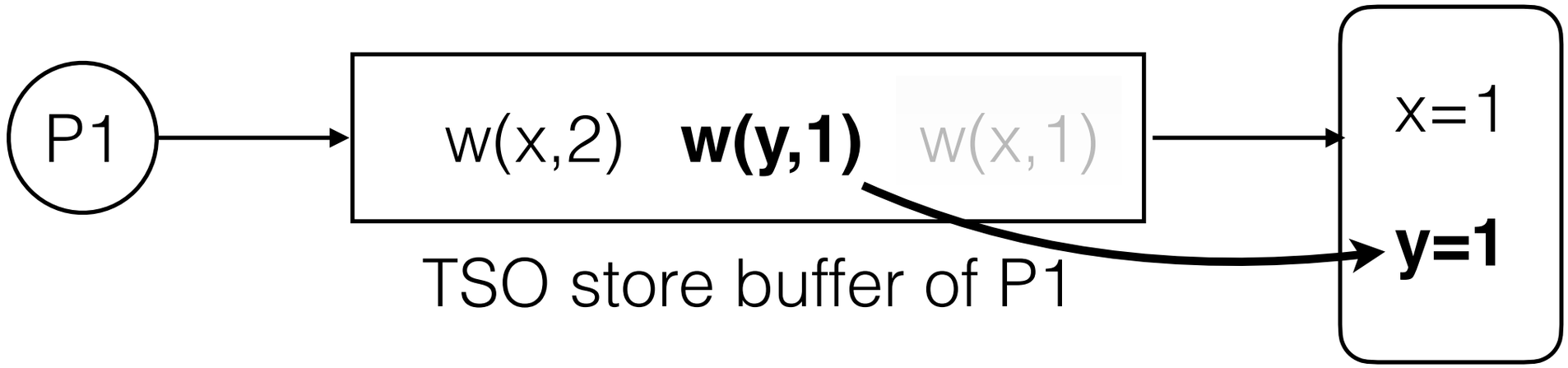


TSO store buffer of P1

An example of TSO program

x=y=0

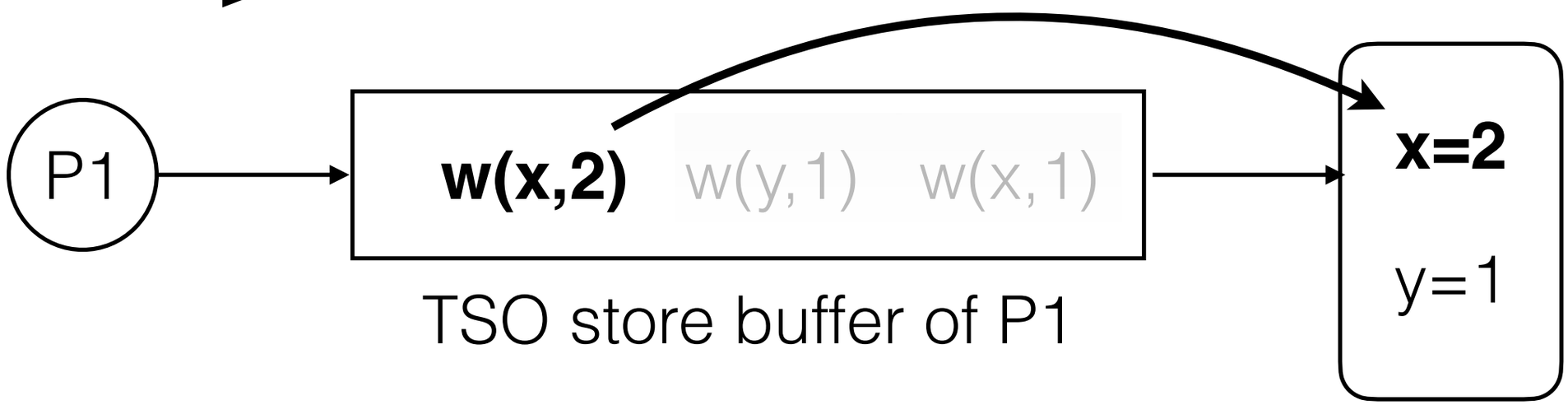
P1	P2
w(x,1)	r(x,2)
w(y,1)	r(y,0)
w(x,2)	



An example of TSO program

x=y=0

P1	P2
w(x,1)	r(x,2)
w(y,1)	r(y,0)
w(x,2)	

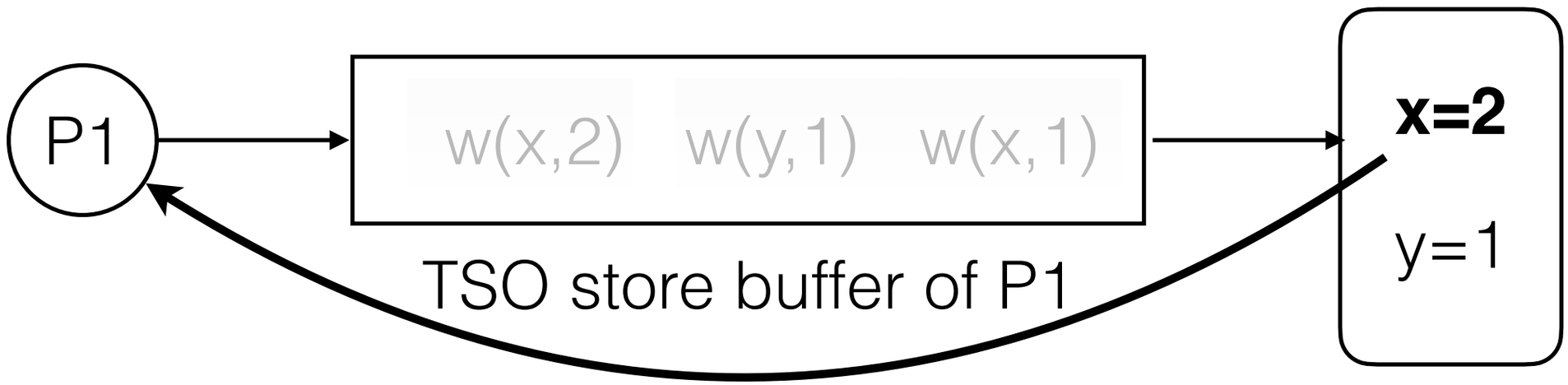


TSO store buffer of P1

An example of TSO program

x=y=0

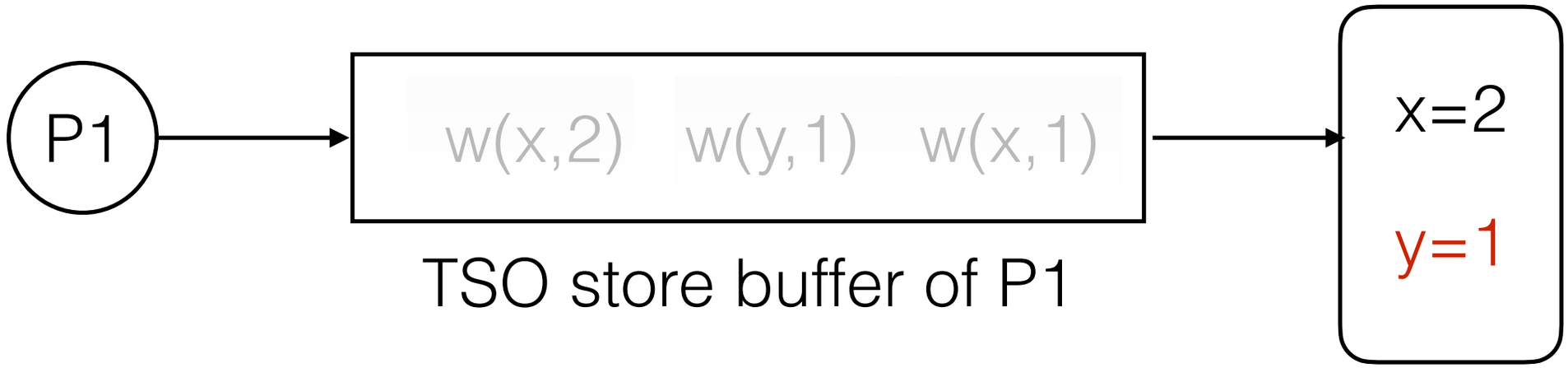
P1	P2
w(x,1)	**r(x,2)**
w(y,1)	r(y,0)
w(x,2)	



An example of TSO program

x=y=0

P1	P2
w(x,1)	r(x,2)
w(y,1)	r(y,0)
w(x,2)	

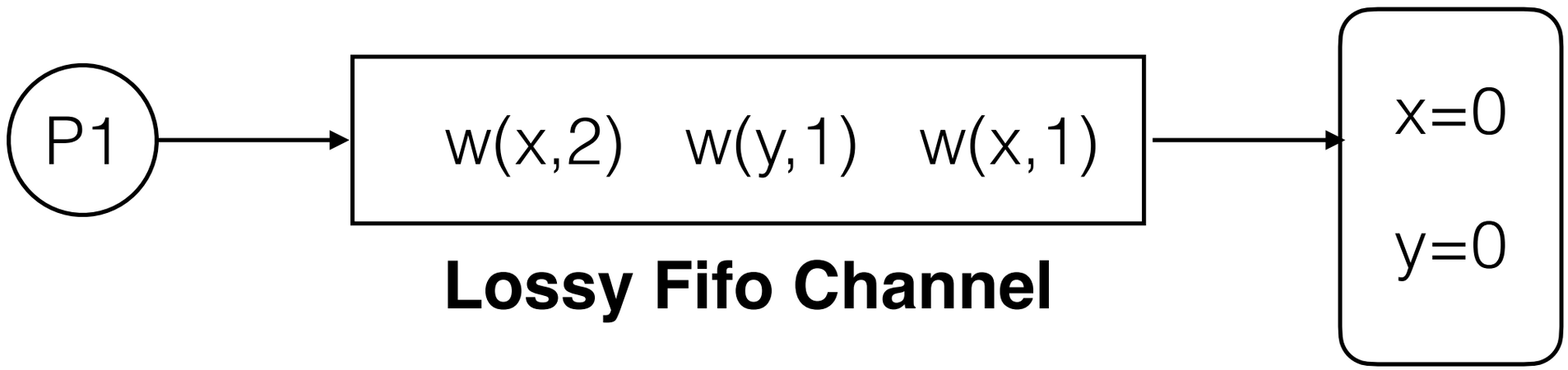


Deadlock under the TSO semantics

TSO Store Buffers —> Lossy Channels ?

x=y=0

P1	P2
w(x,1)	r(x,2)
w(y,1)	r(y,0)
w(x,2)	

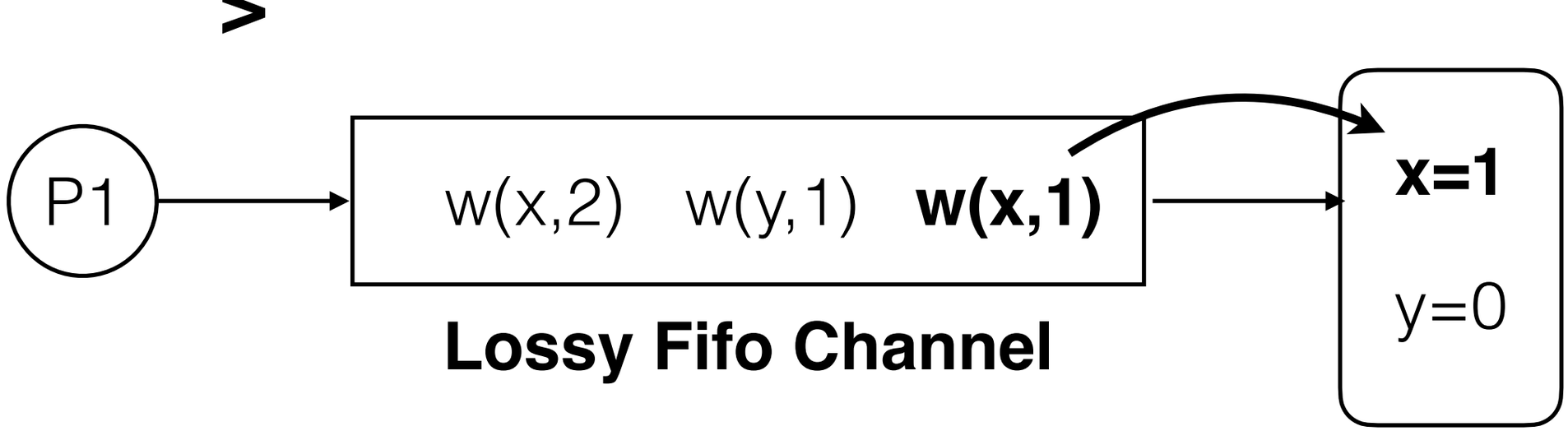


Lossy Fifo Channel

TSO Store Buffers —> Lossy Channels ?

x=y=0

P1	P2
w(x,1)	> r(x,2)
w(y,1)	r(y,0)
w(x,2)	
>	



TSO Store Buffers —> Lossy Channels ?

x=y=0

P1

w(x,1)

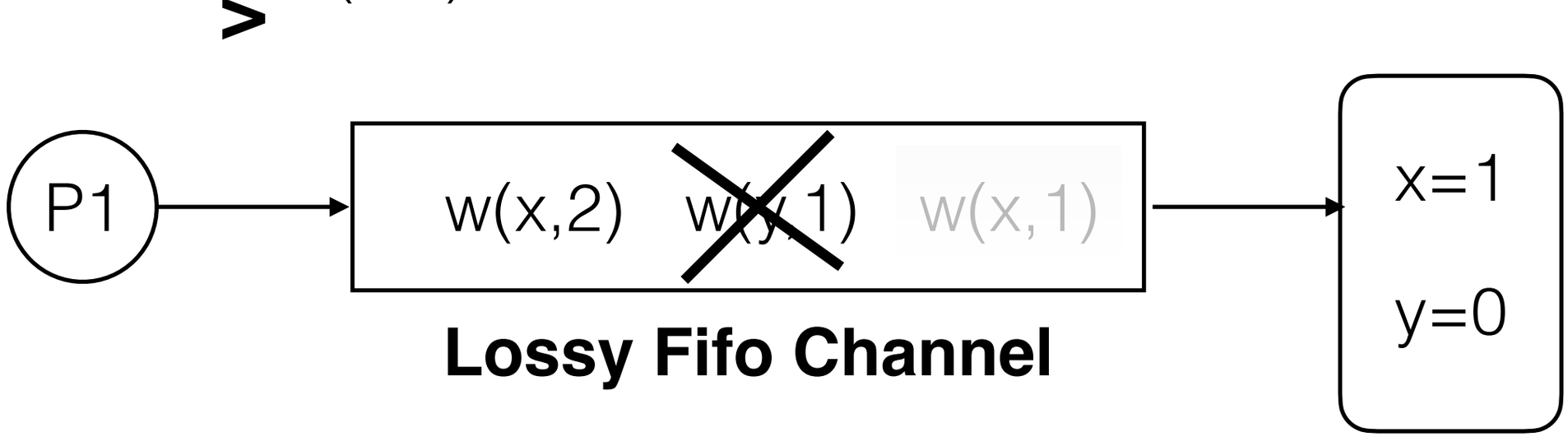
w(y,1)

w(x,2)

P2

r(x,2)

r(y,0)

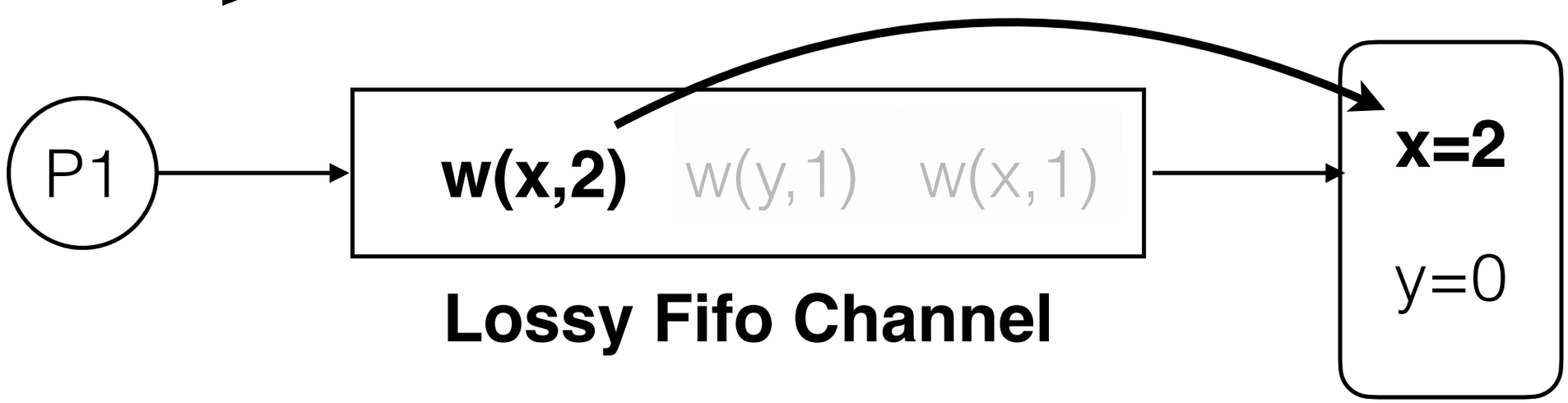


Lossy Fifo Channel

TSO Store Buffers —> Lossy Channels ?

x=y=0

P1	P2
w(x,1)	r(x,2)
w(y,1)	r(y,0)
w(x,2)	



TSO Store Buffers —> Lossy Channels ?

x=y=0

P1

w(x,1)

w(y,1)

w(x,2)

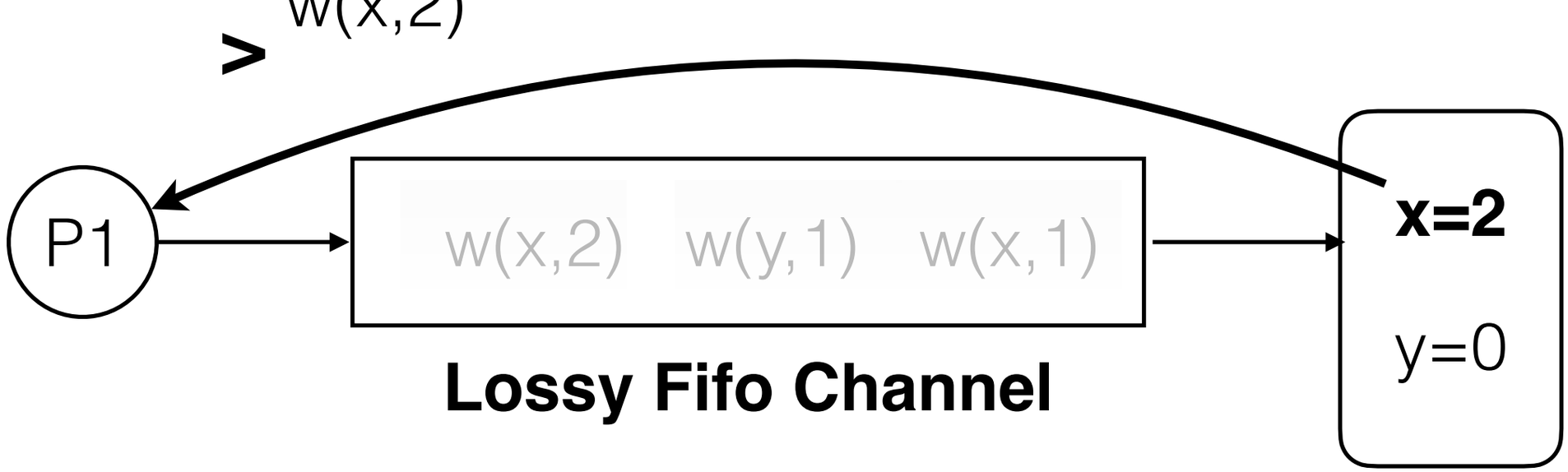
>

P2

r(x,2)

>

r(y,0)



Lossy Fifo Channel

TSO Store Buffers —> Lossy Channels ?

x=y=0

P1

w(x,1)

w(y,1)

w(x,2)

>

P2

r(x,2)

r(y,0)

>

Unsound simulation of TSO!

Store Memory Snapshots

x=y=0

P1

> w(x,1)

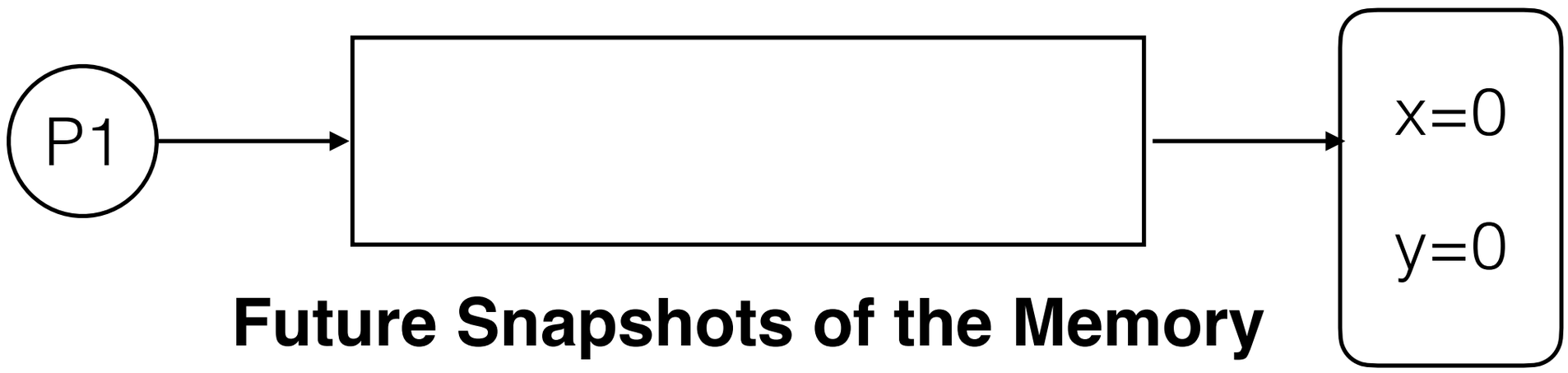
w(y,1)

w(x,2)

P2

> r(x,2)

r(y,0)



Future Snapshots of the Memory

Store Memory Snapshots

x=y=0

P1

w(x,1)

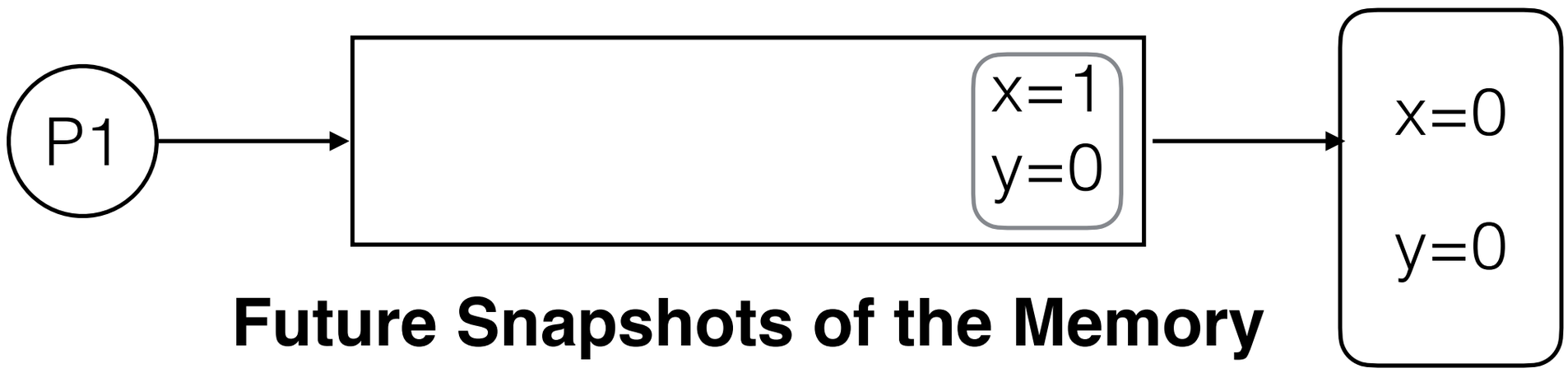
w(y,1)

w(x,2)

P2

r(x,2)

r(y,0)



Future Snapshots of the Memory

Store Memory Snapshots

x=y=0

P1

w(x,1)

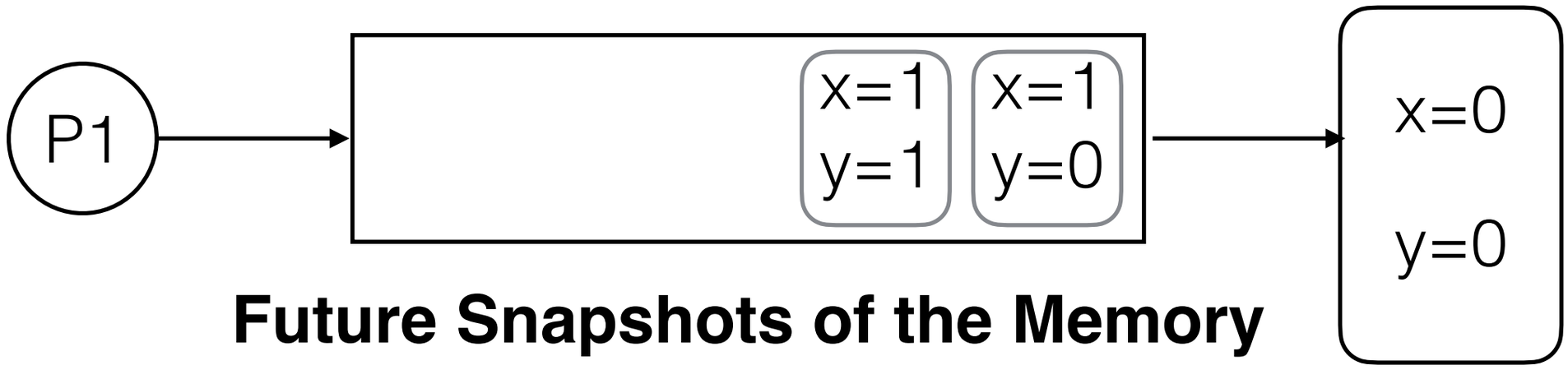
w(y,1)

w(x,2)

P2

r(x,2)

r(y,0)



Future Snapshots of the Memory

Store Memory Snapshots

x=y=0

P1

w(x,1)

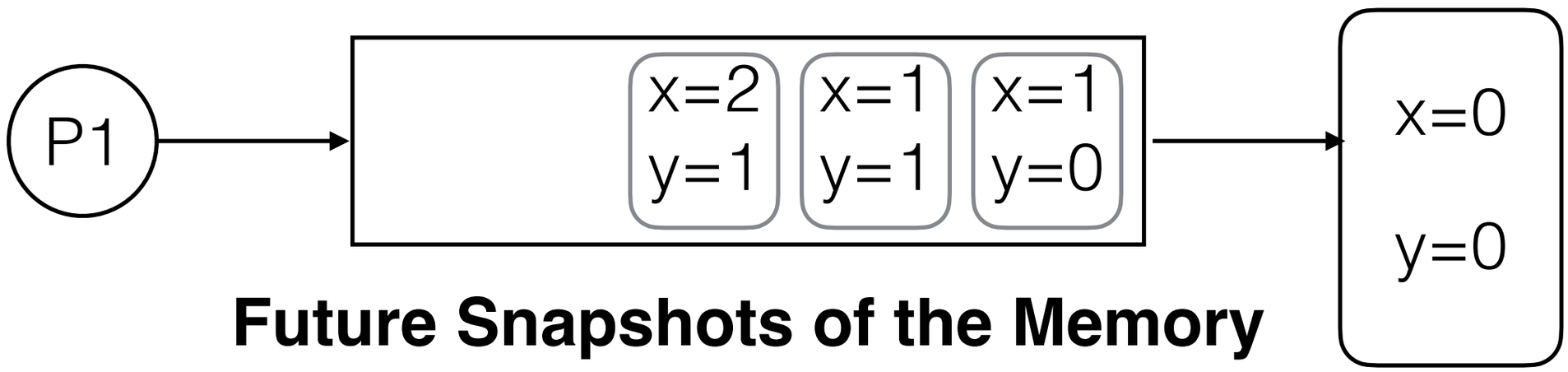
w(y,1)

w(x,2)

P2

r(x,2)

r(y,0)



Future Snapshots of the Memory

Store Memory Snapshots

x=y=0

P1

w(x,1)

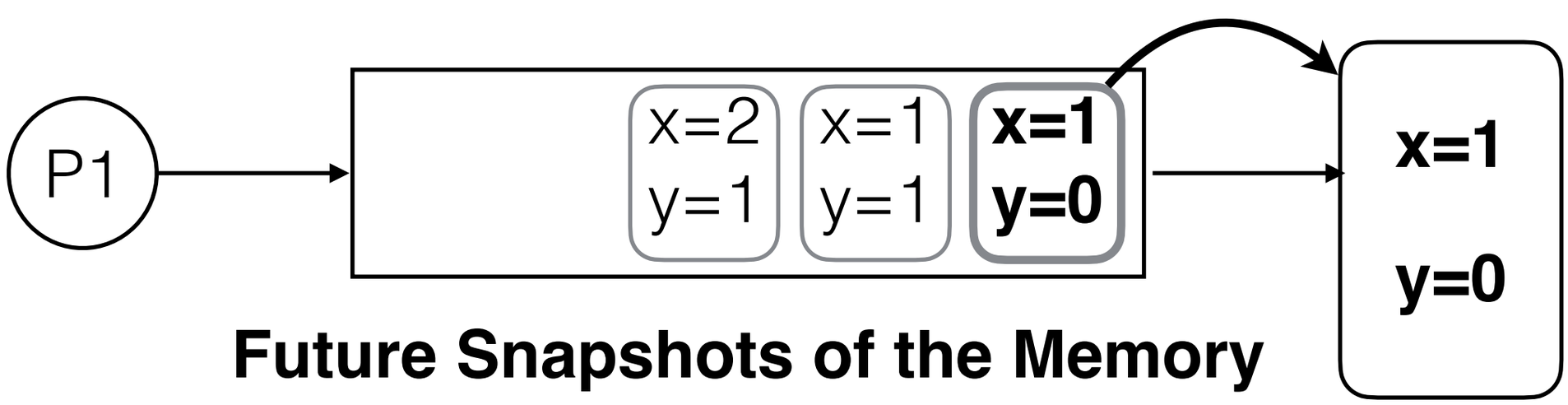
w(y,1)

w(x,2)

P2

r(x,2)

r(y,0)



Future Snapshots of the Memory

Store Memory Snapshots with Losses

x=y=0

P1

w(x,1)

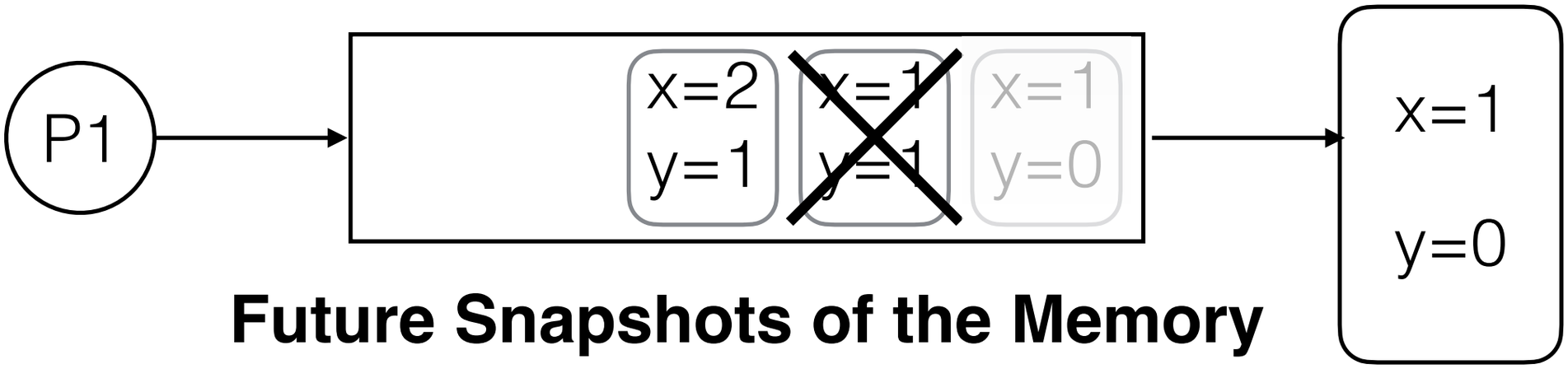
w(y,1)

w(x,2)

P2

r(x,2)

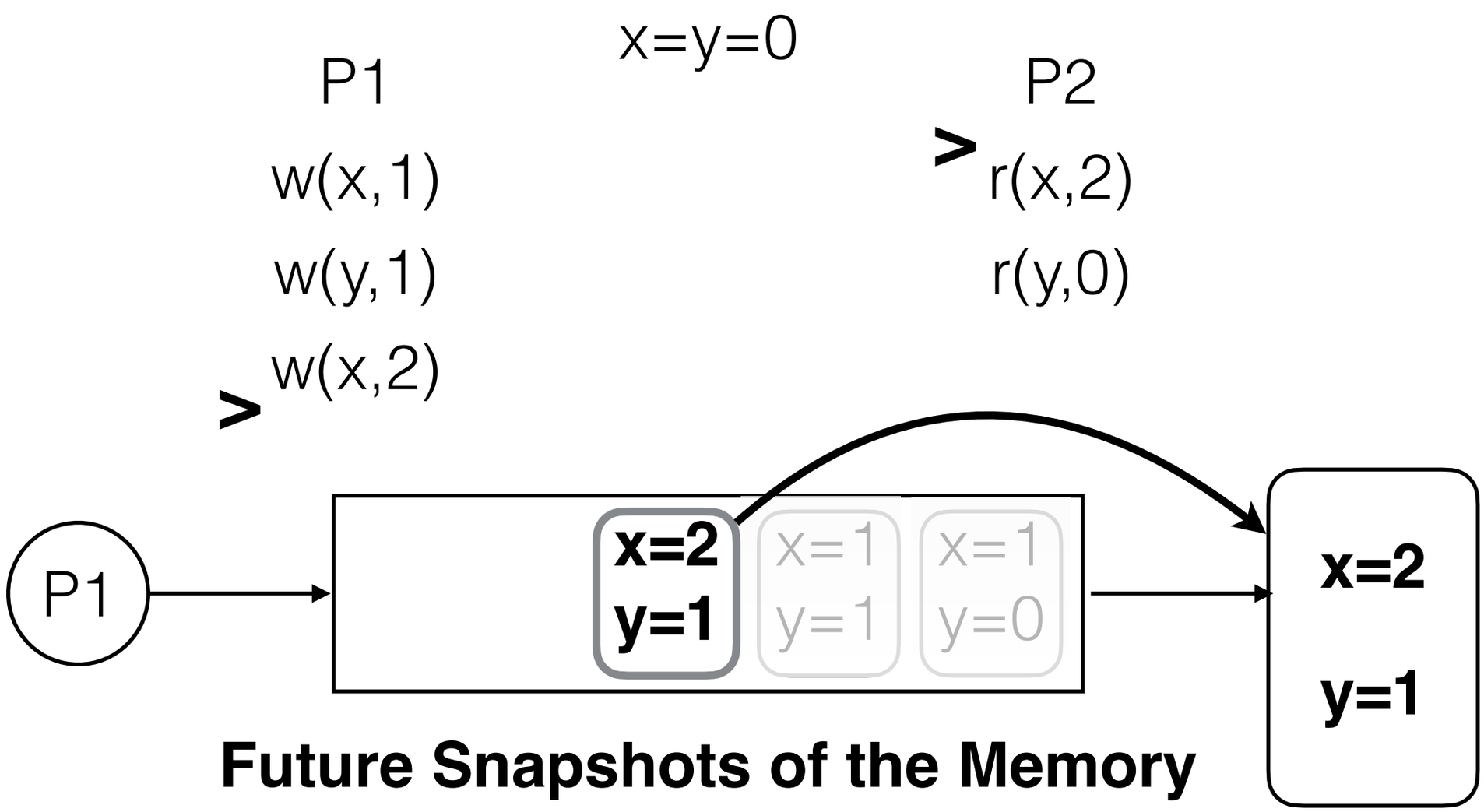
r(y,0)



Future Snapshots of the Memory

+ Lossyness

Store Memory Snapshots with Losses

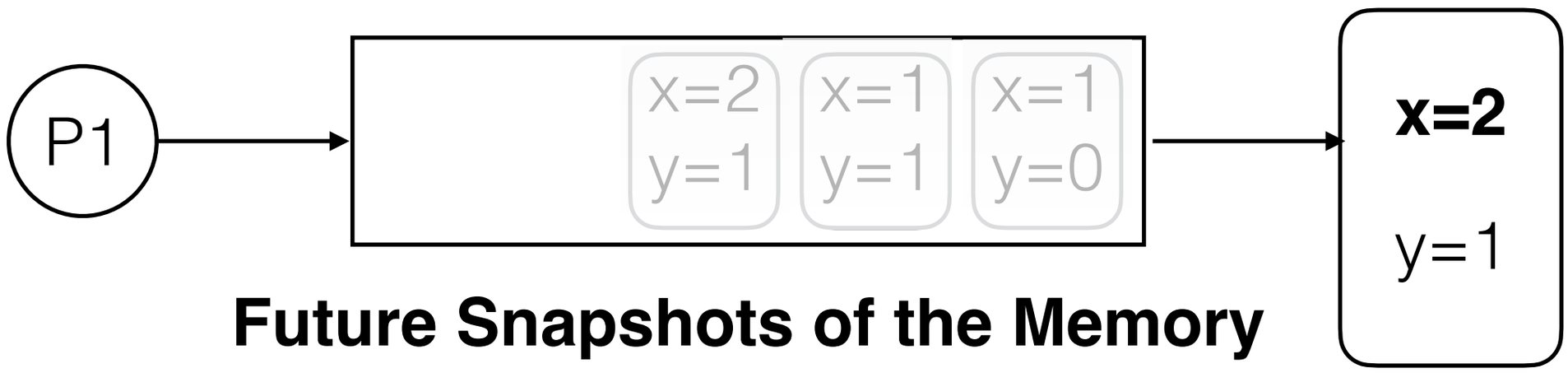


+ Lossyness

Store Memory Snapshots with Losses

x=y=0

P1	P2
w(x,1)	**r(x,2)**
w(y,1)	r(y,0)
w(x,2)	



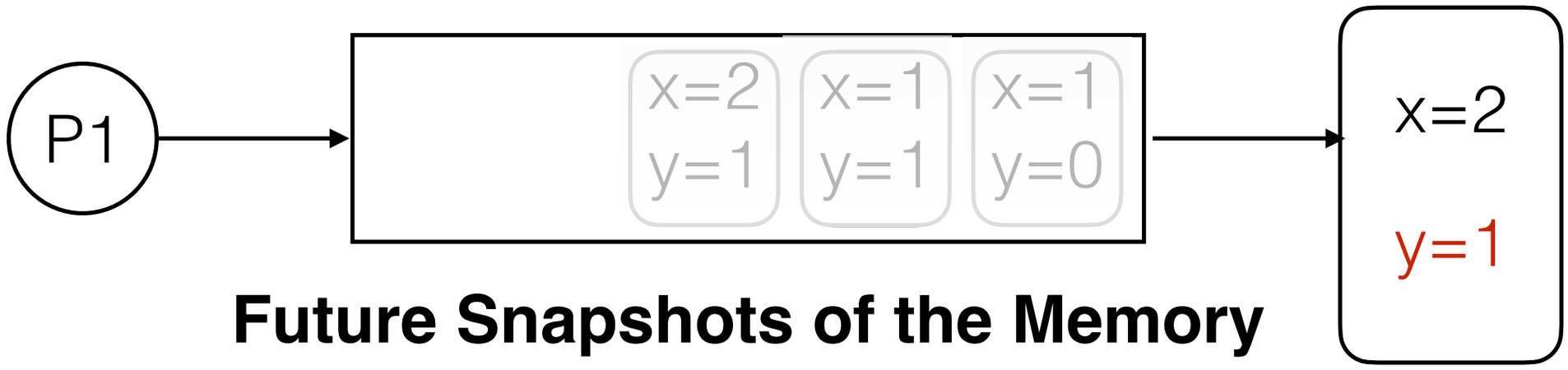
Future Snapshots of the Memory

+ Lossyness

Store Memory Snapshots with Losses

x=y=0

P1	P2
w(x,1)	r(x,2)
w(y,1)	r(y,0)
w(x,2)	



Future Snapshots of the Memory

+ Lossyness

Valid Simulation of TSO

From TSO to Lossy Channel Systems

- 1-channel machine per process + composition

From TSO to Lossy Channel Systems

- 1-channel machine per process + composition

- **Each process:**
  - write: puts a new memory state at the tail of the channel
  - read: checks the channel, then the memory
  - memory update: moves the head of the channel to the memory

From TSO to Lossy Channel Systems

- 1-channel machine per process + composition

- **Each process:**
  - write: puts a new memory state at the tail of the channel
  - read: checks the channel, then the memory
  - memory update: moves the head of the channel to the memory

Problem: Interferences between processes ?
Processes must agree on the same order of memory updates

From TSO to Lossy Channel Systems

- 1-channel machine per process + composition

- **Each process:**
  - write: puts a new memory state at the tail of the channel
  - read: checks the channel, then the memory
  - memory update: moves the head of the channel to the memory

Problem: Interferences between processes ?
Processes must agree on the same order of memory updates

- **guesses writes by other processes**; put them in the channel

- **Validation of the guesses by composition:**
  - transitions are labelled by **write operations + process id**
  - machines are **synchronized** on these actions

Finite number of processes

Reachability for TSO programs

[Atig, B., Burckhardt, Musuvathi, 2010]

Thm: The control state reachability problem under TSO is reducible to the reachability problem in lossy channel systems, and vice-versa.

Reachability for TSO programs

[Atig, B., Burckhardt, Musuvathi, 2010]

Thm: The control state reachability problem under TSO is reducible to the reachability problem in lossy channel systems, and vice-versa.

Coro: The control state reachability problem under TSO is decidable, and it is non primitive recursive.

using [Abdulla & Jonsson1993, Abdulla et al. 1996, Finkel & Schnoebelen 2001, Schnoebelen 2001]

Well …

The complexity is high!

Well …

The complexity is high!

… but this is **not the** main/only **problem**

Well …

The complexity is high!

… but this is **not the** main/only **problem**

The **proposed encoding** of TSO programs as LCS's

- **Is not practical**:

  it requires handling **memory snapshots**

- **Can not be extended to the parametric case**

  it manipulates **process id's**

Well …

The complexity is high!

… but this is **not the** main/only **problem**

The **proposed encoding** of TSO programs as LCS's

- **Is not practical**:

it requires handling **memory snapshots**

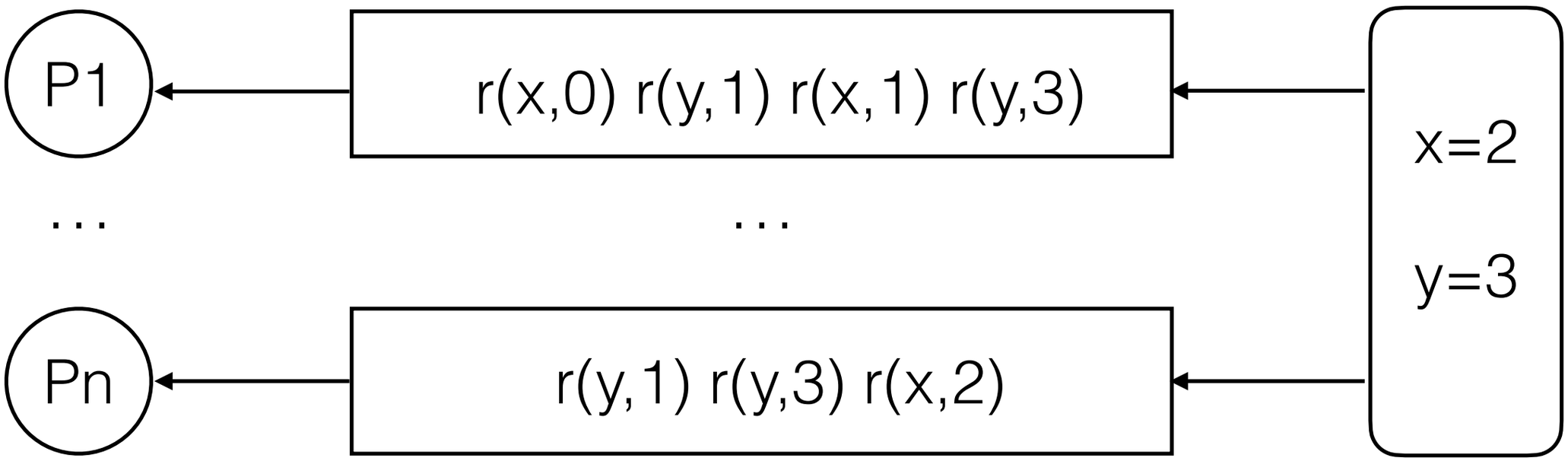
- **Can not be extended to the parametric case**

it manipulates **process id's**

=> We need to change our angle of view…

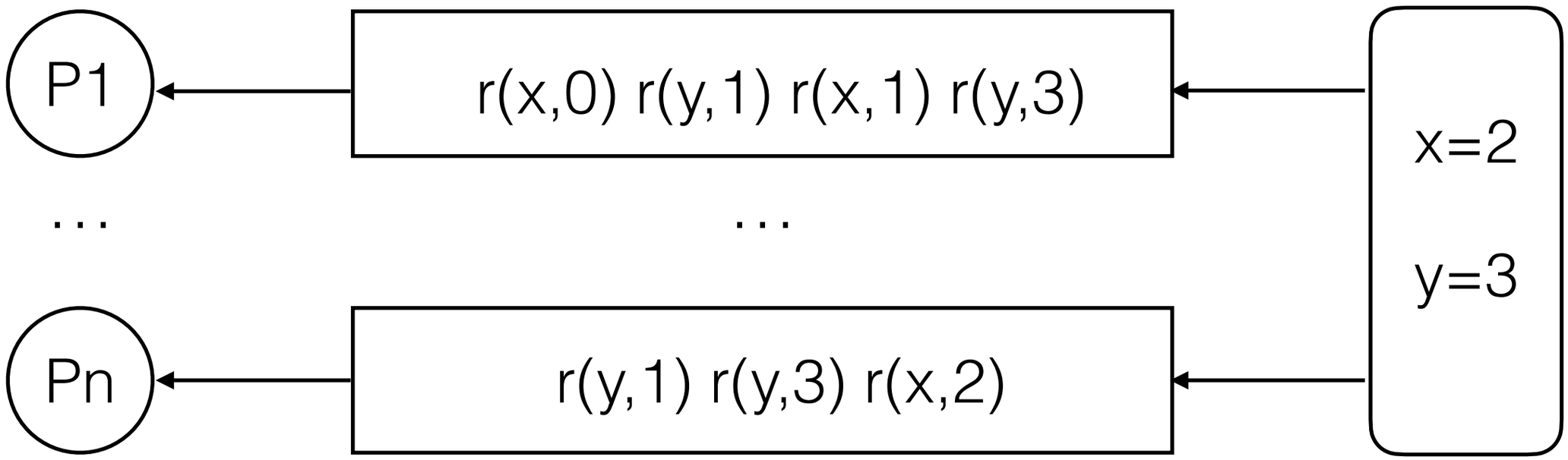
Dual TSO [Abdulla, Atig, B, Ngo, 2016]

- **Store Buffers —> Load Buffers**
- Writes **immediately update** the Memory
- Reads are **sent by the memory** to processes
- Reads **can be skipped** by processes (Load Buffers are **lossy**)



Dual TSO

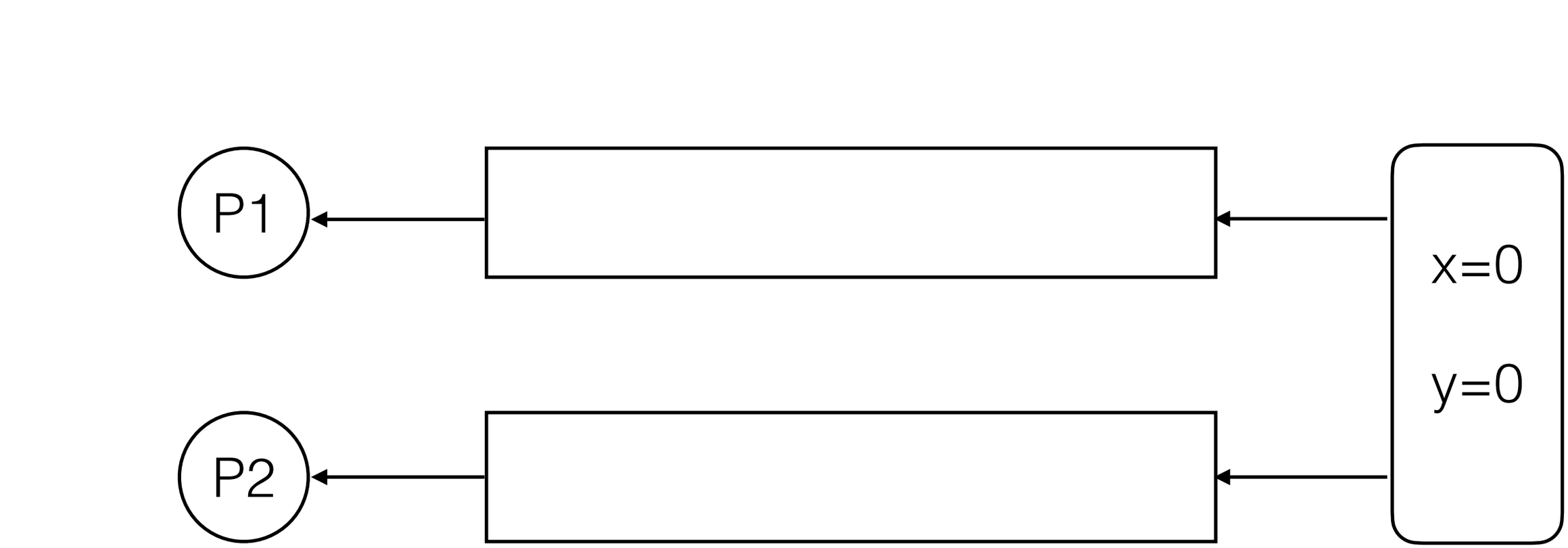
- **Store Buffers —> Load Buffers**
- Writes **immediately update** the Memory
- Reads are **sent by the memory** to processes
- Reads **can be skipped** by processes (Load Buffers are **lossy**)
- => **One** sequence of memory updates (order of writes)
- => Buffers contain **expected reads** by processes
- => Buffers represent a **“(sub)history”** of the memory updates



Dual TSO: Semantics

P1

P2



Dual TSO: Semantics

P1

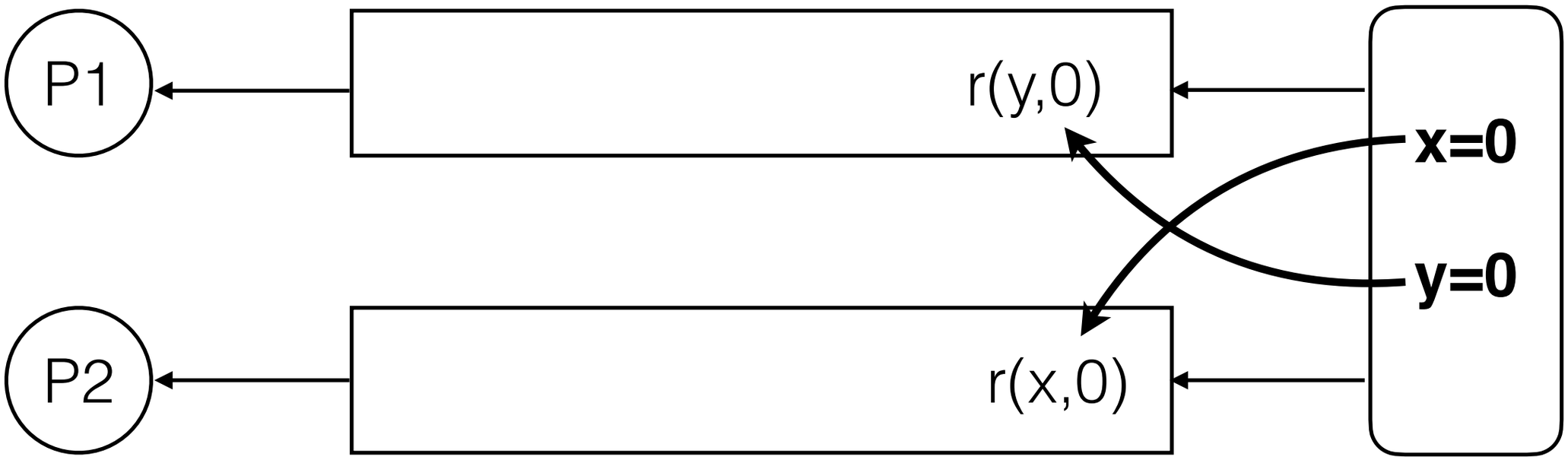
> w(x,1)

r(y,0)

P2

> w(y,1)

r(x,0)



Dual TSO: Semantics

P1

w(x,1)

>

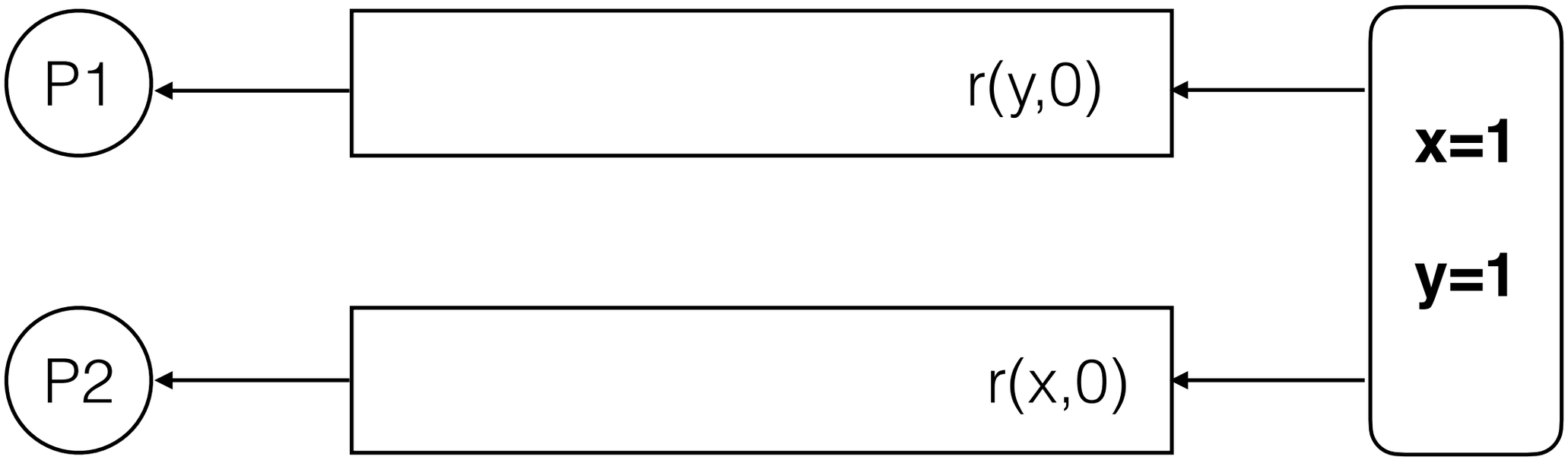
r(y,0)

P2

w(y,1)

>

r(x,0)



Dual TSO: Semantics

P1

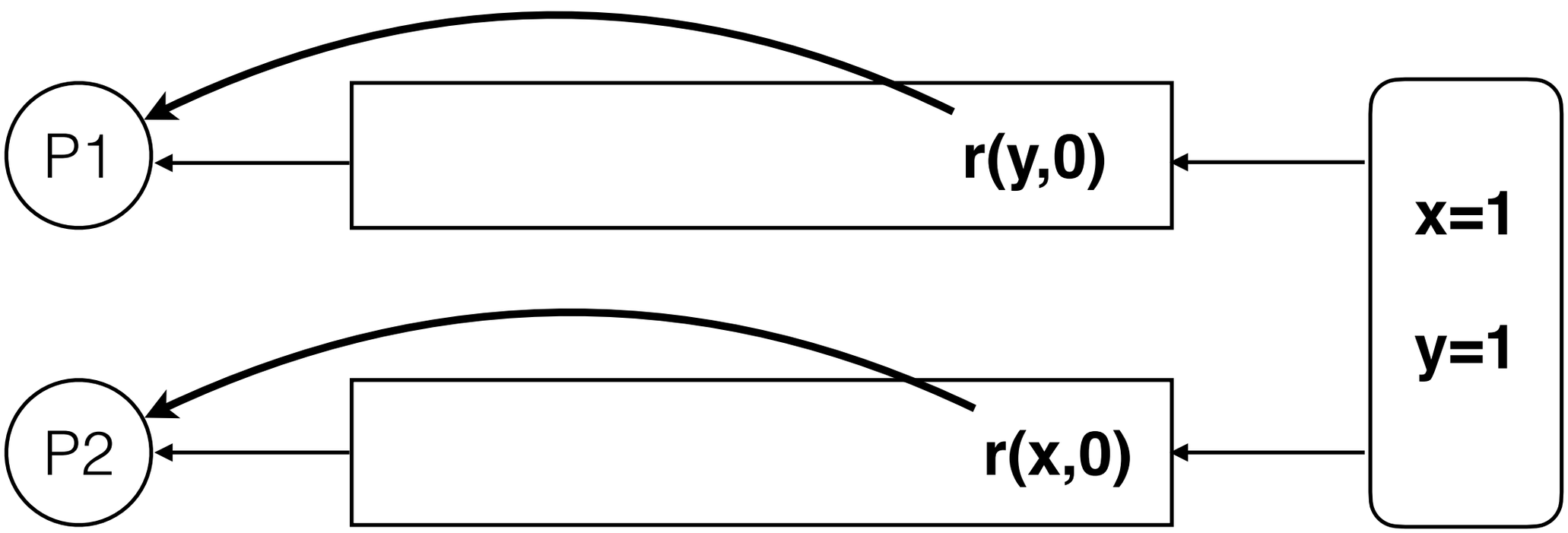
w(x,1)

r(y,0)

P2

w(y,1)

r(x,0)



Dual TSO ~ TSO

Thm: The Dual TSO semantics is equivalent to the TSO semantics with respect to the reachability problem.

Comparing the two encodings

Dual TSO:

- No memory snapshot
- No reference to Process Id's
- Applicable to Parametric Verification
- Implementable verification algorithm

Robustness against Weak Consistency

Given

- An application program P
- A consistency model M1 and a **weaker** model M2

Check if

[P](M) = [P](M')

The sets of visible behaviors of P
under M and M' are equal

Robustness against Weak Consistency

Given

- An application program P
- A consistency model M1 and a **weaker** model M2

Check if

[P](M) = [P](M')

The sets of visible behaviors of P
under M and M' are equal

=> Preservation of safety properties:

Given

- A Safety property Σ
- An abstraction P# of P, i.e., [P](M) subset of [P#](M)

$$\frac{[P\#](M) \models \Sigma \qquad [P\#](M) = [P\#](M')}{[P](M') \models \Sigma}$$

Checking Robustness against TSO

What is observable?

Checking Robustness against TSO

What is observable?

- Reachable memory states

Checking Robustness against TSO

What is observable?

- Reachable memory states
  - **=> solving reachability under TSO**
  - Decidable problem, but highly complex

    [Atig, B., Burckhardt, Musuvathi, POPL'10]

Checking Robustness against TSO

What is observable?

- Reachable memory states
  - **=> solving reachability under TSO**
  - Decidable problem, but highly complex

    [Atig, B., Burckhardt, Musuvathi, POPL'10]

- Traces of computations (po + read-from + write-order)
  - SC computation iff HB (= trace + cf) is acyclic

Traces[SC](P) = Traces[TSO](P)?

Checking Robustness against TSO

What is observable?

- Reachable memory states
  - **=> solving reachability under TSO**
  - Decidable problem, but highly complex

    [Atig, B., Burckhardt, Musuvathi, POPL'10]

- Traces of computations (po + read-from + write-order)
  - SC computation iff HB (= trace + cf) is acyclic

Traces[SC](P) = Traces[TSO](P)?

- Checking if a single computation is SC is possible
- How to verify that all computations are SC ?

Checking Robustness against TSO

What is observable?

- Reachable memory states
  - **=> solving reachability under TSO**
  - Decidable problem, but highly complex

    [Atig, B., Burckhardt, Musuvathi, POPL'10]

- Traces of computations (po + read-from + write-order)
  - SC computation iff HB (= trace + cf) is acyclic

Traces[SC](P) = Traces[TSO](P)?

  - Reduction to **reachability under SC !**
  - (P/EXP)SPACE-complete (for **fixed/arbitrary** nb. of FSM's)

    [B., Derevenetc, Meyer, ESOP'13]

TSO: An SC violation

TSO: An SC violation

P1

w(x,1)

r(y,0)

P2

w(y,1)

r(x,0)

I[w(x,1)]

TSO: An SC violation

P1

w(x,1)

r(y,0)

P2

w(y,1)

r(x,0)

I[w(x,1)]

TSO: An SC violation

P1

w(x,1)

r(y,0)

P2

w(y,1)

r(x,0)

l[w(x,1)] r(y,0) w(y,1)

TSO: An SC violation

P1

w(x,1)

r(y,0)

P2

w(y,1)

r(x,0)

l[w(x,1)] r(y,0) w(y,1) r(x,0)

TSO: An SC violation

P1

w(x,1)

r(y,0)

>

P2

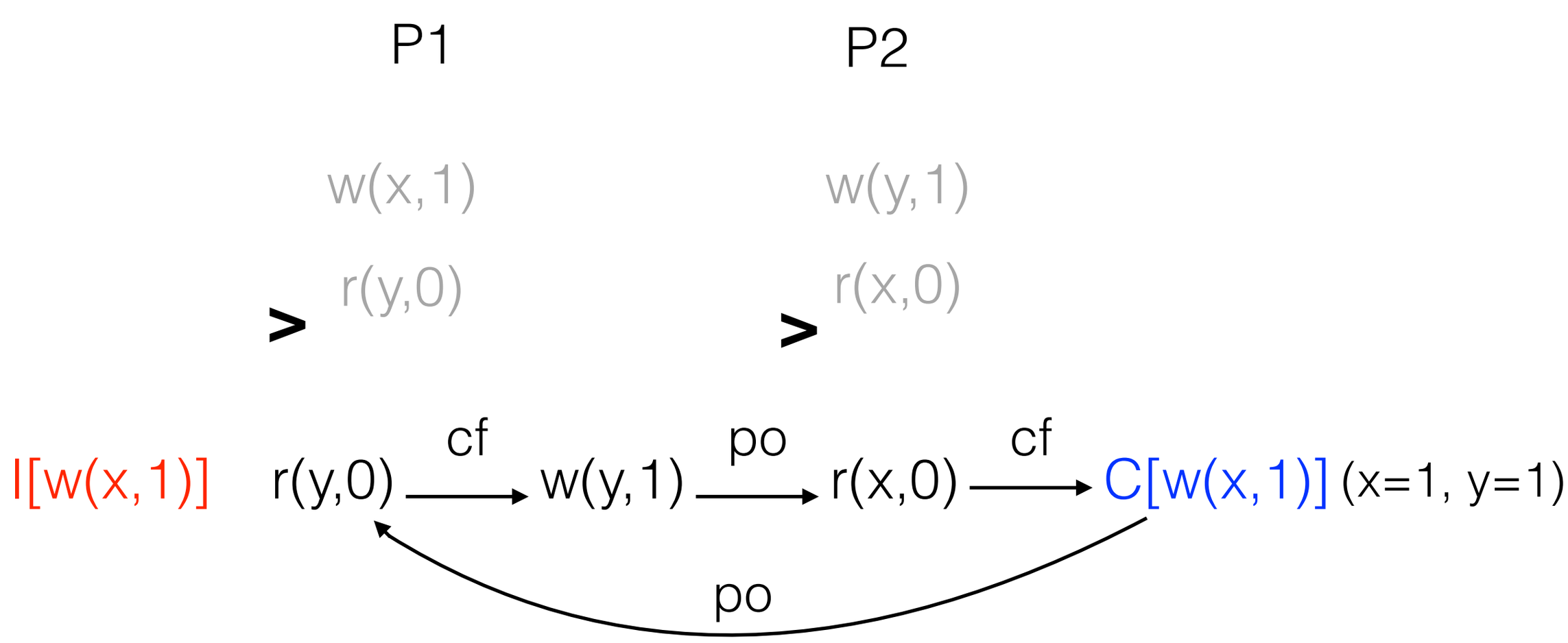
w(y,1)

r(x,0)

>

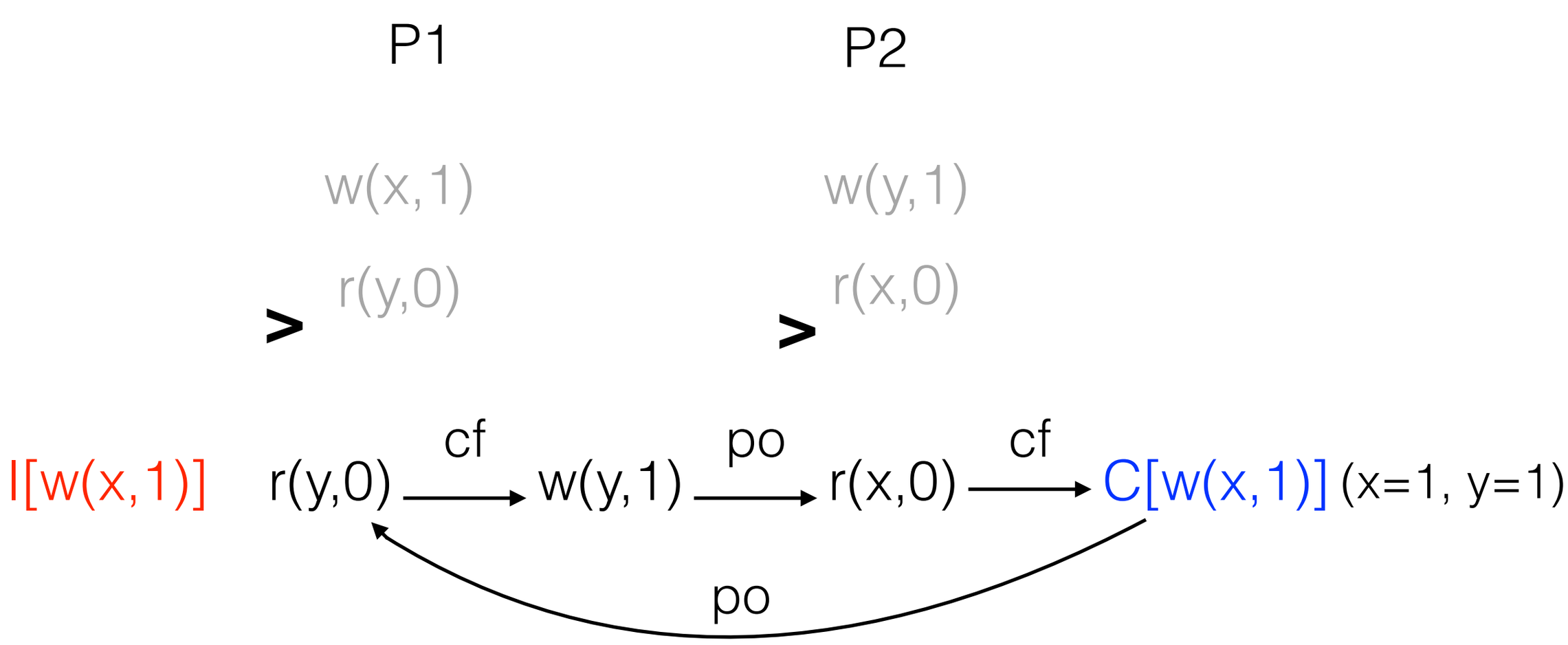
I[w(x,1)] r(y,0) w(y,1) r(x,0) C[w(x,1)] (x=1, y=1)

TSO: An SC violation



- **Minimal (*borderline*) SC violation** (in the # of order relaxations)
- **Only one process is delaying writes** (here P1) — pair of write-read
- **Bad pattern**: Cycle characterized by a pair **W** and **R** of one process
  - **W** (and subsequent writes) are delayed to let **R** read some old value
  - **W** and **R** are conflicting

TSO: An SC violation



- **Minimal (*borderline*) SC violation** (in the # of order relaxations)
- **Only one process is delaying writes** (here P1) — pair of write-read
- **Bad pattern**: Cycle characterized by a pair **W** and **R** of one process
  - **W** (and subsequent writes) are delayed to let **R** read some old value
  - **W** and **R** are conflicting

COMPLETE

Checking Robustness against TSO

Traces[SC](P) = Traces[TSO](P)?

Intrumentation of P —> P’

For each pair W, R

- Guess the occurrence of W to delay
- Check the existence of a **hb** path reaching **R**
- If yes, go to a special state **F**

P is trace-robust iff F is not reachable in [P’](SC)

- Reduction to **reachability under SC !**
- (P/EXP)SPACE-complete (for **fixed/arbitrary** nb. of FSM’s)

  [B., Derevenetc, Meyer, ESOP’13]

Robustness against Weak Consistency

Transactional models

Serializability (SER), Snapshot Isolation,
Causal Consistency (CC), Prefix Consistency, etc.

- SER vs CC [Beillahi, B., Enea, CONCUR'19]

- SER vs SI [Beillahi, B., Enea, CAV'19]

- SI vs PC and PC vs CC [Beillahi, B., Enea, ESOP'21]

Robustness against Weak Consistency

Transactional models

Serializability (SER), Snapshot Isolation,
Causal Consistency (CC), Prefix Consistency, etc.

- SER vs CC [Beillahi, B., Enea, CONCUR'19]

- SER vs SI [Beillahi, B., Enea, CAV'19]

- SI vs PC and PC vs CC [Beillahi, B., Enea, ESOP'21]

- Characterize what separate the two models
- Notion of borderline/minimal violation
- Finite number of patterns to track

- Efficient and precise static analysis techniques

Conclusion

- Safety verification: Decidability / complexity still open in many cases
- When decidable, the complexity is high
- Verifying and enforcing robustness is an important problem
- Efficient upper/under approximate methods have been developed

Future work

- Liveness still needs to be investigated

[Abdulla, Atig, Godbole, Krishna, Vahanwala, 2023]

- Efficient verification techniques are needed (e.g., PO techniques …)

[B., Enea, Roman-Calvo, 2023]

- General frameworks for specifying consistency levels
- Composing systems with different consistency levels
- Tuning consistency levels by need